

The fontspec package

WILL ROBERTSON and KHALED HOSNY
will.robertson@latex-project.org

2010/09/19 v2.1

Contents

1 History	3	7.4 Different features for different font shapes .	14
2 Introduction	3	7.5 Different features for different font sizes . .	15
2.1 About this manual . .	3	8 Font independent options	16
2.2 Acknowledgements . .	4	8.1 Colour	16
3 Package loading and options	4	8.2 Scale	17
3.1 Maths fonts adjustments	5	8.3 Interword space	17
3.2 Configuration	5	8.4 Post-punctuation space	18
3.3 Warnings	5	8.5 The hyphenation character	18
I General font selection	5	8.6 Optical font sizes . . .	19
4 Font selection	6	II OpenType	20
4.1 By font name	6	9 Introduction	20
4.2 By file name	6	9.1 How to select font features	21
5 Default font families	8	10 Complete listing of OpenType font features	21
6 New commands to select font families	8	10.1 Ligatures	21
6.1 More control over font shape selection	10	10.2 Letters	22
6.2 Math(s) fonts	11	10.3 Numbers	23
6.3 Miscellaneous font selecting details	12	10.4 Contextuals	24
7 Selecting font features	12	10.5 Vertical Position	24
7.1 Default settings	12	10.6 Fractions	26
7.2 Changing the currently selected features	13	10.7 Stylistic Set variations	26
7.3 Priority of feature selection	13	10.8 Character Variants . .	27
		10.9 Alternates	27
		10.10 Style	27
		10.11 Diacritics	29
		10.12 Kerning	29
		10.13 Font transformations .	30

10.14	Annotation	30	17	Renaming existing features & options	45
10.15	CJK shape	31	18	Programming details	46
10.16	Character width	31			
10.17	Vertical typesetting . . .	33	VI	The patching/improvement of L^AT_EX 2_ε and other pack- ages	47
10.18	OpenType scripts and languages	33	19	Inner emphasis	48
III	LuaT_EX-only font fea- tures	34	20	Unicode footnote symbols	48
11	OpenType font feature files	34	21	Verbatim	48
IV	Fonts and features with X_YT_EX	35	22	Discretionary hyphenation: \-	48
12	X _Y T _E X-only font features	35	VII	fontspec.sty	49
12.1	Mapping	35	23	Implementation	49
12.2	Letter spacing	35	23.1	Bits and pieces	49
12.3	Different font technolo- gies: AAT and ICU	38	23.2	Error/warning messages	50
12.4	Optical font sizes	38	23.3	Option processing	54
13	Mac OS X's AAT fonts	38	23.4	Packages	54
13.1	Ligatures	39	23.5	Encodings	54
13.2	Letters	39	23.6	User commands	55
13.3	Numbers	39	23.7	Programmer's interface	60
13.4	Contextuals	39	23.8	expl3 interface for font loading	64
13.5	Vertical position	40	23.9	Internal macros	64
13.6	Fractions	40	23.10	keyval definitions	80
13.7	Variants	40	23.11	Italic small caps	99
13.8	Alternates	42	23.12	Selecting maths fonts . .	100
13.9	Style	42	23.13	Finishing up	104
13.10	CJK shape	42	VIII	fontspec.lua	105
13.11	Character width	42	IX	fontspec-patches.sty	108
13.12	Vertical typesetting . . .	42	23.14	Unicode footnote sym- bols	108
13.13	Diacritics	43	23.15	Emph	108
13.14	Annotation	43	23.16	\-	108
14	AAT & Multiple Master font axes	44	23.17	Verbatims	108
V	Programming interface	44	X	fontspec.cfg	111
15	Defining new features	44			
16	Going behind fontspec's back	45			

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 document is typeset with $\text{pdf}\LaTeX$ using pre-compiled examples that have been generated by either X_{\TeX} or $\text{Lua}\TeX$. You may regenerate the examples by removing the `doc/` subdirectory and typesetting the manual with the following invocation:

```
pdflatex -shell-escape fontspec.dtx
```

Note that many of the examples use fonts that are not included in \TeX Live or MiKTeX , and some of them are non-free fonts that must be purchased.

I'd like to reduce the number of non-free fonts used in this manual. 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.)

2.2 Acknowledgements

This package couldn't be possible without the early and continued support the author of X_YTeX, Jonathan Kew. When I started this package, he steered me many times in the right direction.

I've had great feedback over the years on feature requests, documentation queries, bug reports, font suggestions, and so on from lots of people all around the world. Many thanks to you all.

Thanks to David Perry for numerous documentation improvements and contributing the text for one of the sections of this manual.

Special thanks to Khaled Hosny, who had been the driving force behind the support for LuaTeX, ultimately leading to version 2.0 of the package.

3 Package loading and options

For basic use, no package options are required:

```
\usepackage{fontspec}
```

xunicode Ross Moore's xunicode package is now automatically loaded for users of both X_YTeX and LuaTeX. This package provides backwards compatibility with TeX's methods for accessing extra characters and accents (for example, `\%`, `\$`, `\textbullet`, `\u`, and so on), plus many more Unicode characters. **Warning:** introduced in v2.1, this is a backwards incompatible change to previous versions of fontspec. This change was necessary in order to provide consistent support for users of X_YTeX and LuaTeX. I'm not aware of any issues this may cause but please let me know if you run into problems.

X_YTeX users only The xltextra package adds some minor extra features to X_YTeX, including, via the metalogo package, the `\XeTeX` macro to typeset the X_YTeX logo. While this package was previously recommended, it serves a much smaller rôle nowadays and generally will not be required.

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.

`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 \XeTeX , the `polyglossia` package is recommended instead as a modern replacement for `babel`.

3.1 Maths fonts adjustments

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

If you find that `fontspec` 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 44](#) and [Section 17](#)), this should be done by creating your own `fontspec.cfg` file, which will be automatically loaded if it is found by \XeTeX or \LuaTeX . Either place it in the same folder as the main document for isolated cases, or in a location that \XeTeX or \LuaTeX searches by default; e.g. in \MacTeX : `~/Library/texmf/tex/latex/`. The package option `[no-config]` will suppress this behaviour under all circumstances.

3.3 Warnings

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

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 $\langle font\ feature \rangle = \langle option \rangle$ lists; these are described in later:

- For general font features, see [Section 8 on page 16](#)
- For OpenType fonts, see [Part II on page 20](#)
- For XeTeX-only general font features, see [Part IV on page 35](#)
- For LuaTeX-only general font features, see [Part III on page 34](#)
- For features for AAT fonts in XeTeX, see [Section 13 on page 38](#)

4 Font selection

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

4.1 By font name

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

XeTeX and LuaTeX also allow fonts to be loaded by file name instead of font name. When you have a very large collection of fonts, you will sometimes not wish to have them all installed in your system’s font directories. In this case, it is more convenient to load them from a different location on your disk. This technique is also necessary in XeTeX when loading OpenType fonts that are present within your TeX distribution, such as `/usr/local/texlive/2010/texmf-dist/fonts/opentype/public`. Fonts in such locations are visible to XeTeX but cannot be loaded by font name, only file name; LuaTeX does not have this restriction.

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.

X_YTeX & Mac users only: Note that X_YTeX can only select fonts in this way with the xdvipdfmx driver, but X_YTeX with the xdv2pdf driver can only select system-installed fonts by font name and not file name. The xdvipdfmx driver is default for X_YTeX, so this is only a problem if you wish to explicitly use the xdv2pdf driver.

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

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

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

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

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

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

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

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

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

Note that X_YTeX and 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’:

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

```
\setmainfont{TeX Gyre Bonum}
\setsansfont[Scale=MatchLowercase]{Latin Modern Sans}
\setmonofont[Scale=MatchLowercase]{Inconsolata}

Pack my box with five dozen liquor jugs \rmfamily Pack my box with five dozen liquor jugs\par
Pack my box with five dozen liquor jugs \sffamily Pack my box with five dozen liquor jugs\par
Pack my box with five dozen liquor jugs \ttfamily Pack my box with five dozen liquor jugs
```

```
\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 16](#), including methods for automatic scaling.

6 New commands to select font families

```
\newfontfamily \⟨font-switch⟩ [⟨font features⟩] {⟨font name⟩}
\newfontface \⟨font-switch⟩ [⟨font features⟩] {⟨font name⟩}
```

For cases when a specific font with a specific feature set is going to be re-used many times in a document, it is inefficient to keep calling `\fontspec` for every use. While the `\fontspec` 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

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>

the font inside its argument (i.e., the same behaviour as `\emph`) 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 being 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 39](#).

Comment for advanced users: The commands defined by `\newfontface` and `\newfontfamily` include their encoding information, so even if the document is set to use a legacy T_EX encoding, such commands will still work correctly. For example,

```
\documentclass{article}
\usepackage{fontspec}
\newfontfamily\unicodefont{Lucida Grande}
\usepackage{mathpazo}
\usepackage[T1]{fontenc}
\begin{document}
A legacy \TeX\ font. {\unicodefont A unicode font.}
\end{document}
```

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>

6.1 More control over font shape selection

BoldFont = **
ItalicFont = **
BoldItalicFont = **
SlantedFont = **
BoldSlantedFont = **
SmallCapsFont = **

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 among. The **BoldFont** and **ItalicFont** features are provided for these situations. If only one of these is used, the bold italic font is requested as the default from the *new* font. 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 [Section 4.2 on page 6](#).) 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 L^AT_EX 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}
```

For most modern fonts that have small caps as a font feature, this level of control isn't generally necessary, but you may still occasionally find font families in which the small caps are in a separate font.

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. When an OpenType font is selected for `SmallCapsFont`, the small caps font feature is *not* automatically enabled. In this case, users should write instead

```
\fontspec[
  SmallCapsFont={...},
  SmallCapsFeatures={Letters=SmallCaps},
]{...}
```

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.¹)

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.

<pre>\setmathrm [⟨font features⟩] {⟨font name⟩} \setmathsf [⟨font features⟩] {⟨font name⟩} \setmathtt [⟨font features⟩] {⟨font name⟩} \setboldmathrm [⟨font features⟩] {⟨font name⟩}</pre>
--

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 the commands above (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

¹Speaking 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}}`

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:

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

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

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

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

7 Selecting font features

The commands discussed so far such as `\fontspec` each take an optional argument for accessing the font features of the requested font. Commands are provided to set default features to be applied for all fonts, and even to change the features that a font is presently loaded with. Different font shapes can be loaded with separate features, and different features can even be selected for different sizes that the font appears in. This section discusses these options.

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.

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

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

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

	<code>\fontspec[Numbers={Proportional,OldStyle}]</code>
	<code>{TeX Gyre Adventor}</code>
	‘In 1842, 999 people sailed 97 miles in
	13 boats. In 1923, 111 people sailed 54
	miles in 56 boats.’ <code>\bigskip</code>
‘In 1842, 999 people sailed 97 miles in 13 boats. In	<code>{\addfontfeatures{Numbers={Monospaced,Lining}}}</code>
1923, 111 people sailed 54 miles in 56 boats.’	<code>\begin{tabular}{@{} cccc @{}}</code>
	<code>Year & People & Miles & Boats \</code>
	<code>\hline 1842 & 999 & 75 & 13 \</code>
	<code>1923 & 111 & 54 & 56</code>
	<code>\end{tabular}}</code>

‘In 1842, 999 people sailed 97 miles in 13 boats. In 1923, 111 people sailed 54 miles in 56 boats.’

Year	People	Miles	Boats
1842	999	75	13
1923	111	54	56

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, 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.

Example 7: Features for, say, just italics.

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

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>

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.

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.

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</i> ITALIC SMALL CAPS	<code>\itshape Italic {\scshape Italic Small Caps}\</code>
Bold BOLD SMALL CAPS	<code>\upshape\bfseries Bold {\scshape Bold Small Caps}\</code>
<i>Bold Italic</i> BOLD ITALIC SMALL CAPS	<code>\itshape Bold Italic {\scshape Bold Italic Small Caps}</code>

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>

7.5 Different features for different font sizes

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

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

It takes a comma separated list of braced, comma separated lists of features for each size range. Each sub-list must contain the Size option to declare the size range, and optionally Font to change the font based on size. Other (regular) fontspec features that are added are used on top of the font features that would be used anyway. 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 19](#).

To be precise, the Size sub-feature accepts arguments in the form shown in Ta-

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	$s = X$

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}
```

[ble 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 Colour

Color (or Colour), also shown in [Section 7.1 on page 12](#) and elsewhere, uses font specifications to set the colour of the text. The colour is defined as a triplet of two-digit Hex RGB values, with optionally another value for the transparency (where 00 is completely transparent and FF is opaque.) Transparency is supported by Lua¹TeX and by Xe²TeX with the xdv2pdf driver (Mac OS X only); Xe³TeX with the xdvipdfmx driver does not support this feature.

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

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>

```
\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 colour using the Opacity feature:

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

8.2 Scale

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

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

As well as a numerical argument, Scale feature also accepts options MatchLowercase and MatchUppercase, which will scale the font being selected to match the current default roman font to either the height of the lowercase or uppercase letters, respectively; 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.

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 feature is provided, which takes either a single scaling factor to scale the default value, or a triplet of comma-separated values to scale the

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

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

Example 14: Scaling the default post-punctuation space.

<p>Letters, Words. Sentences.</p> <p>Letters, Words. Sentences.</p> <p>Letters, Words. Sentences.</p>	<pre> \nonfrenchspacing \fontspec{TeX Gyre Schola} Letters, Words. Sentences. \fontspec{PunctuationSpace=2}{TeX Gyre Schola} Letters, Words. Sentences. \fontspec{PunctuationSpace=0}{TeX Gyre Schola} Letters, Words. Sentences. </pre>
---	--

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, T_EX 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.

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.

Example 15: Explicitly choosing the hyphenation character.		
<div>EXAMPLE HYPHENATION</div>	<pre>\def\text{\fbox{\parbox{1.55cm}{% EXAMPLE HYPHENATION% }}\quad\quad\null\par\bigskip} \fontspec{Linux Libertine} \addfontfeature{HyphenChar=None} \text \addfontfeature{HyphenChar={+}} \text</pre>	
<div>EXAMPLE HYPHEN+ ATION</div>		
Example 16: A demonstration of automatic optical size selection.		
Automatic optical size	<pre>\fontspec{Latin Modern Roman} Automatic optical size \scalebox{0.4}{\Huge Automatic optical size}</pre>	\\
Automatic optical size		

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 as in Example 16, in which we've scaled down some large text in order to be able to compare the difference for equivalent font sizes.

The `OpticalSize` option may be used to specify a different optical size. With `OpticalSize` set to zero, no optical size font substitution is performed, as shown in Example 17.

The `SizeFeatures` feature (Section 7.5 on page 15) 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}}
]{Latin Modern Roman}

```

Example 17: Optical size substitution is suppressed when set to zero.

	<code>\fontspec[OpticalSize=0]{Latin Modern Roman 5 Regular}</code>
	Latin Modern optical sizes <code>\\</code>
	<code>\fontspec[OpticalSize=0]{Latin Modern Roman 8 Regular}</code>
	Latin Modern optical sizes <code>\\</code>
Latin Modern optical sizes	<code>\fontspec[OpticalSize=0]{Latin Modern Roman 12 Regular}</code>
Latin Modern optical sizes	Latin Modern optical sizes <code>\\</code>
Latin Modern optical sizes	<code>\fontspec[OpticalSize=0]{Latin Modern Roman 17 Regular}</code>
Latin Modern optical sizes	Latin Modern optical sizes

Part II

OpenType

9 Introduction

OpenType fonts (and other ‘smart’ font technologies such as AAT and Graphite) can change the appearance of text in many different ways. These changes are referred to as features. When the user applies a feature — for example, small capitals — to a run of text, the code inside the font makes appropriate adjustments and small capitals appear in place of lowercase letters. However, the use of such features does not affect the underlying text. In our small caps example, the lowercase letters are still stored in the document; only the appearance has been changed by the OpenType feature. This makes it possible to search and copy text without difficulty. If the user selected a different font that does not support small caps, the ‘plain’ lowercase letters would appear instead.

Some OpenType features are required to support particular scripts, and these features are often applied automatically. The scripts used in India, for example, often require that characters be reshaped and reordered after they are typed by the user, in order to display them in the traditional ways that readers expect. Other features can be applied to support a particular language. The Junicon font for medievalists uses by default the Old English shape of the letter thorn, while in modern Icelandic thorn has a more rounded shape. If a user tags some text as being in Icelandic, Junicon will automatically change to the Icelandic shape through an OpenType feature that localizes the shapes of letters.

A very large group of OpenType features is designed to support high quality typography in Latin, Greek, Cyrillic and other standard scripts. Examples of some font features have already been shown in previous sections; the complete set of OpenType font features supported by fontspec is described below in [Section 10](#).

The OpenType specification provides four-letter codes (e.g., `smcp` for small capitals) for each feature. The four-letter codes are given below along with the fontspec names for various features, for the benefit of people who are already familiar with OpenType. You can ignore the codes if they don’t mean anything to you.

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
	Historic	hlig
	TeX	tlig/trep

* This feature is activated by default.

9.1 How to select font features

Font features are selected by a series of $\langle feature \rangle = \langle option \rangle$ selections. Features are (usually) grouped logically; for example, all font features relating to ligatures are accessed by writing `Ligatures={...}` with the appropriate argument(s), which could be `TeX`, `Rare`, etc., as shown below in [Section 10.1](#).

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

If a feature or an option is requested that the font does not have, a warning is given in the console output. As mentioned in [Section 3.3 on page 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 fonts² is shown in [Example 18](#).

Note the additional features accessed with `Ligatures=TeX`. These are not actually real OpenType features, but additions provided by `luaotfload` (i.e., \LuaTeX only) to emulate \TeX ’s behaviour for `ASCII` input of curly quotes and punctuation. In \XeTeX this is achieved with the `Mapping` feature (see [Section 12.1 on page 35](#)) but for consistency `Ligatures=TeX` will perform the same function as `Mapping=tex-text`.

²<http://www.linuxlibertine.org/>

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=Historic}{strict} \test{Ligatures=Rare}{wurtzite} \test{Ligatures=NoCommon}{firefly} </pre>
wurtzite	→	wurtzite	
firefly	→	firefly	

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

10.2 Letters

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

Petite caps are smaller than small caps. SmallCaps and PetiteCaps turn lowercase letters into the smaller caps letters, whereas the Uppercase . . . options turn the *capital* letters into the smaller caps (good, *e.g.*, for applying to already uppercase acronyms like ‘NASA’). 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.³ It is designed select various uppercase forms for glyphs such as accents and dashes, such as shown in Example 20; note the raised position of the hyphen to better match the surrounding letters.

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

Example 19: Small caps from lowercase or uppercase letters.

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

Example 20: An example of the Uppercase option of the Letters feature.	
	<code>\fontspec{Linux Libertine}</code>
UPPER-CASE example	UPPER-CASE example \
UPPER-CASE example	<code>\addfontfeature{Letters=Uppercase}</code>
	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

The Kerning feature also contains an Uppercase option, which adds a small amount of spacing in between letters (see [Section 10.12 on page 29](#)).

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 13](#). 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 (with tag anum) maps regular numerals to their Arabic script or Persian equivalents based on the current Language setting (see [Section 10.18 on page 33](#)), shown in [Example 22](#) using the Zar fonts⁴. This option is based on a LuaTeX feature of the luaotfload package, not an OpenType feature. (Thus, this feature is unavailable in XeTeX.)

⁴http://wiki.irmug.org/index.php/X_Series_2

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

Example 22: An example of number remapping to Arabic or Persian. (LuaTeX only.)		
• ١٢٣٤٥٦٧٨٩	\fontspec[Script=Arabic,Numbers=Arabic]{XB Zar}	
• ١٢٣٤٥٦٧٨٩	{\addfontfeature{Language=Arabic}	
	0123456789} \\\	
	{\addfontfeature{Language=Parsi}	
	0123456789}	

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

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

10.4 Contextuals

This feature refers to substitutions of glyphs that vary ‘contextually’ by their relative position in a word or string of characters; features such as 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.

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 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 `xltxtra` for $\text{X}\text{\LaTeX}$) redefines the `\textsubscript` and `\textsuperscript` commands to use the above font features

Example 23: An example of the Swashes option of the Contextuals feature.		
<i>Without Contextual Swashes</i>	\fontspec{Warnock Pro} \itshape	
	Without Contextual Swashes \\\	
<i>With Contextual Swashes; cf. W C S</i>	\fontspec[Contextuals=Swash]{Warnock Pro}	
	With Contextual Swashes; cf. W C S	

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

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

Example 24: The VerticalPosition feature. Note that the Ordinal option can be quite unreliable, as the results here demonstrate.

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

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

Feature	Option	Tag
Fractions	= On	frac
	Alternate	afrc

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>

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 Stylistic Set variations

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.

Two demonstrations from the Junicon font⁵ 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 44 for a way to assign names to stylistic sets, which should be done on a per-font basis.

⁵<http://junicode.sf.net>

Example 26: Insular letterforms, as used in medieval Northern Europe, for the Junicon font accessed with the StylisticSet feature.

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

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

ENLARGED Minuscules.	<code>\fontspec{Junicode}</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>
	<code>\textsc{a} \& h \</code>
A & h	<code>\addfontfeature{Alternate=0}</code>
	<code>\textsc{a} \& h</code>

10.8 Character Variants

Similar to the ‘Stylistic Sets’ above, ‘Character Variations’ are selected numerically to adjust the output of (usually) a single character for the particular font. These correspond to the OpenType features `cv01` to `cv99`.

I don’t have a font to demonstrate this with (please suggest one if you know of a free font with this feature!), but the syntax is similar to that above:

```
\fontspec[CharacterVariant={1,3,5}]{...}
```

10.9 Alternates

The Alternate feature (for the raw OpenType feature `alt`) 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 44](#) for a way to assign names to alternates, which must be done on a per-font basis.

10.10 Style

‘Ruby’ refers to a small optical size, used in Japanese typography for annotations. For fonts with multiple `alt` 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.

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

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

Example 29: Example of the Alternate option of the Style feature.

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

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

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

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

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

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

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

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>
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=VerticalKana}}</code>
ようこそ ワカヨタレソ	<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.

Note the difference here between the default and the horizontal style kana in Example 33: the horizontal style is slightly wider.

10.11 Diacritics

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

10.12 Kerning

Specifies how inter-glyph spacing should behave.

As briefly mentioned previously at the end of [Section 10.2 on page 22](#), the Uppercase option will add a small amount of tracking between uppercase letters,

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.

Example 34: Adding extra kerning for uppercase letters. (The difference is usually very small.)

UPPERCASE EXAMPLE
UPPERCASE EXAMPLE

```
\fontspec{Romande ADF Std Bold}
UPPERCASE EXAMPLE \
\addfontfeature{Kerning=Uppercase}
UPPERCASE EXAMPLE
```

Example 35: Artificial font transformations.

ABCxyz *ABCxyz*
ABCxyz **ABCxyz**
ABCxyz **ABCxyz**

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

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

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

seen in Example 34, which uses the Romande fonts⁶ (thanks to Clea F. Rees for the suggestion).

10.13 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.

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.

10.14 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.

⁶<http://arkandis.tuxfamily.org/adffonts.html>

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	
㊦ ㊧ ㊨ ㊩ ㊪ ㊫ ㊬ ㊭ ㊮	
㊯ ㊰ ㊱ ㊲ ㊳ ㊴ ㊵ ㊶ ㊷	
㊸ ㊹ ㊺ ㊻ ㊼ ㊽ ㊾ ㊿	
1. 2. 3. 4. 5. 6. 7. 8. 9.	
	\fontspec{Hiragino Maru Gothic Pro} 1 2 3 4 5 6 7 8 9 \def\x#1{\{\{\addfontfeature{Annotation=#1} 1 2 3 4 5 6 7 8 9 }\} \x0\x1\x2\x3\x4\x5\x6\x7\x8\x9

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

10.15 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.

10.16 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.

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>

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.

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

Example 39: Numbers can be compressed significantly.

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

The same situation occurs with numbers, which are provided in increasingly illegible compressed forms seen in Example 39.

10.17 Vertical typesetting

TODO!

10.18 OpenType scripts and languages

Fonts that include glyphs for various scripts and languages may contain different font features for the different character sets and languages they support, and different font features may behave differently depending on the script or language chosen. When multilingual fonts are used, it is important to select which language they are being used for, and more importantly what script is being used.

The ‘script’ refers to the alphabet in used; for example, both English and French use the Latin script. Similarly, the Arabic script can be used to write in both the Arabic and Persian languages.

The Script and Language features are used to designate this information. The possible options are tabulated in Table 13 on page 35 and Table 14 on page 36, 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 automatically selected by fontspec before all others and, if Xe_{La}TeX is being used, will specifically select the ICU renderer for this font, as described in Section 12.3 on page 38.

10.18.1 Script and Language examples

In the following examples, the Code2000 font⁷ is used to typeset the input with and without the OpenType Script applied. The text is only rendered correctly in the second case; many examples of incorrect diacritic spacing as well as a lack of contextual ligatures and rearrangement can be seen. Thanks to Jonathan Kew, Yves Codet and Gildas Hamel for their contributions towards these examples.

⁷<http://www.code2000.net/>

Example 40: An example of various Scripts and Languages.

العربي	الْعَرَبِيّ	
हिन्दी	हिन्दी	
লেখ	লেখ	
મર્યાદા-સૂચક નિવેદન	मर्यादा-सूचक निवेदन	<code>\testfeature{Script=Arabic}{\arabictext}</code>
மமீடிசை பாரம்பரிய	மமீடிசை பாரம்பரிய	<code>\testfeature{Script=Devanagari}{\devanagaritext}</code>
આદિ સચુ જુગાદિ સચુ	आदि सचु जुगादि सचु	<code>\testfeature{Script=Bengali}{\bengalitext}</code>
தமிழ் துடே	தமிழ் துடே	<code>\testfeature{Script=Gujarati}{\gujaratitext}</code>
תּוֹרָה	תּוֹרָה	<code>\testfeature{Script=Malayalam}{\malayalamtext}</code>
cáp số mõi	cáp số mõi	<code>\testfeature{Script=Gurmukhi}{\gurmukhitext}</code>
		<code>\testfeature{Script=Tamil}{\tamiltext}</code>
		<code>\testfeature{Script=Hebrew}{\hebrewtext}</code>
		<code>\def\examplefont{Doulos SIL}</code>
		<code>\testfeature{Language=Vietnamese}{\vietnamesetext}</code>

10.18.2 Defining new scripts and languages

`\newfontscript` While the scripts and languages listed in [Table 13](#) and [Table 14](#) are intended to be comprehensive, there may be some missing; alternatively, you might wish to use different names to access scripts/languages that are already listed. Adding scripts and languages can be performed with the `\newfontscript` and `\newfontlanguage` commands. For example,

```
\newfontscript{Arabic}{arab}
\newfontlanguage{Zulu}{ZUL}
```

The first argument is the fontspec name, the second the OpenType tag. The advantage to using these commands rather than `\newfontfeature` (see [Section 15 on page 44](#)) is the error-checking that is performed when the script or language is requested.

Part III

LuaTeX-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 Adobe⁸.

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

⁸http://www.adobe.com/devnet/opentype/afdko/topic_feature_file_syntax.html

A feature file is loaded by passing its name or path to `FeatureFile`; OpenType features defined in the file can then be applied as usual to the font.

For example, to add OpenType mapping of f-ligatures in the 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}
```

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 41.

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

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

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;
```

Example 41: X_YTeX's Mapping feature.

“¡A small amount of—text!”	\fontspec[Mapping=tex-text]{Cochin} “‘!‘A small amount of---text!’”
----------------------------	--

LetterSpace, which takes a numeric argument, shown in Example 42.

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 22).

Example 42: 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
--	---

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 10.18 on page 33](#) 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 44](#) 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_YTeX (due to its pedigree on Mac OS X in the first place) and was the first font format supported by `fontspec`. A number of fonts distributed with Mac OS X are still in the AAT format, such as 'Skia'. Documents that use these fonts should be compiled with X_YL^AT_EX using the `xdv2pdf` driver, as opposed to the default `xdvipdfmx`. E.g.,

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

Example 43: 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>

Mac OS X also supports Multiple Master fonts, which are discussed in [Section 14](#).

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 T_EX trick of splitting up a ligature with an empty brace pair does not work in X_YT_EX; you must use a 0 pt kern or `\hbox` (e.g., `\null`) to split the characters up if you do not want a ligature to be performed (the usual examples for when this might be desired are words like ‘shelffull’).

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 AAT fonts, they may be a combination of Lining or OldStyle and Proportional or Monospaced (the latter is good for tabular material). The synonyms Uppercase and Lowercase are equivalent to Lining and OldStyle, respectively. The differences have been shown previously in [Section 7.2 on page 13](#).

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 43), LineInitial, LineFinal, and Inner (Example 44, 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: A contextual feature for the ‘long s’ can be convenient as the character does not need to be marked up explicitly.

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

Example 45: 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>\</code>
Normal ^{superior} _{inferior} 1 st 2 nd 3 rd 4 th 0 th 8abcde	<code>\fontspec[VerticalPosition=Ordinal]{Skia}</code> 1st 2nd 3rd 4th 0th 8abcde

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 45.

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 46.

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 47.

13.7 Variants

The `Variant` feature takes a single numerical input for choosing different alphabetic shapes. Don’t mind my fancy Example 48 :) I’m just looping through the nine (!) variants of Zapfino.

See Section 15 on page 44 for a way to assign names to variants, which should be done on a per-font basis.

Example 46: Fractions in AAT fonts. The `^^^^2044` glyph is the ‘fraction slash’ that may be typed in Mac OS X with `OPT+SHIFT+1`; not shown literally here due to font constraints.

	<code>\fontspec[Fractions=On]{Skia}</code>
	<code>1{^^^^2044}2 \quad 5{^^^^2044}6 \\ % fraction slash</code>
$\frac{1}{2}$ $\frac{5}{6}$	<code>1/2 \quad 5/6 % regular slash</code>
$\frac{1}{2}$ $\frac{5}{6}$	<code>\fontspec[Fractions=Diagonal]{Skia}</code>
$\frac{13579}{24680}$	<code>13579{^^^^2044}24680 \\ % fraction slash</code>
$\frac{13579}{24680}$	<code>\quad 13579/24680 % regular slash</code>

Example 47: Alternate design of pre-composed fractions.

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

Example 48: 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}}

```

Example 49: Alternate shape selection must be numerical.

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

13.8 Alternates

Selection of Alternates *again* must be done numerically; see Example 49. See [Section 15 on page 44](#) for a way to assign names to alternates, which should be done on a per-font basis.

13.9 Style

The options of the Style feature are defined in AAT as one of the following: Display, Engraved, IlluminatedCaps, Italic, Ruby,⁹ TallCaps, or TitlingCaps.

Typical examples for these features are shown in [Section 10.10](#).

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 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 10.16 on page 31](#) 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!

X₃TeX provides for vertical typesetting simply with the ability to rotate the individual glyphs as a font is used for typesetting, as shown in Example 50.

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 X₃TeX documentation.

⁹‘Ruby’ refers to a small optical size, used in Japanese typography for annotations.

Example 50: Vertical typesetting.

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
```

Example 51: 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>	
	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	

13.13 Diacritics

Diacritics are marks, such as the acute accent or the tilde, applied to letters; they usually indicate a change in pronunciation. In Arabic scripts, diacritics are used to indicate vowels. You may either choose to Show, Hide or Decompose them in AAT fonts. The Hide option is for scripts such as Arabic which may be displayed either with or without vowel markings. E.g., `\fontspec[Diacritics=Hide]{...}`

Some older fonts distributed with Mac OS X included ‘Ø’ *etc.* as shorthand for writing ‘Ø’ under the label of the Diacritics feature. If you come across such fonts, 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 recommend using the proper \LaTeX input conventions for obtaining such characters instead.

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 51) with the following options: Off, Box, RoundedBox, Circle, BlackCircle, Parenthesis, Period, RomanNumerals, Diamond, BlackSquare, BlackRoundSquare, and DoubleCircle.

Example 52: 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	

Example 53: Assigning new AAT features.

	<code>\newAATfeature{Alternate}{HoeflerSwash}{17}{1}</code>
<i>This is XeTeX by Jonathan Kew.</i>	<code>\fontspec[Alternate=HoeflerSwash]{Hoefler Text Italic}</code>
	This is XeTeX by Jonathan Kew.

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 52. Variations along a multiple master font's optical size axis has been shown previously in [Section 8.6 on page 19](#).

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 53.

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

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

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

Here's what it would look like in practise:

Example 54: 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 55: Using raw font features directly.

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

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

`\newfontfeature`

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

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

For example, Zapfino contains the feature 'Avoid d-collisions'. To access it with this package, you could do some like that shown in Example 54

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 L^AT_EX font selection conveniences. The `RawFeature` font feature 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 56.

Example 56: Renaming font features.	
	<code>\aliasfontfeature{ItalicFeatures}{IF}</code>
Roman Letters <i>And Swash</i>	<code>\fontspec[IF = {Alternate=1}]{Hoefler Text}</code>
	Roman Letters \itshape And Swash

Example 57: Renaming font feature options.	
	<code>\aliasfontfeature{VerticalPosition}{Vert Pos}</code>
	<code>\aliasfontfeatureoption{VerticalPosition}{ScientificInferior}{Sci Inf}</code>
	<code>\fontspec[Vert Pos=Sci Inf]{Linux Libertine}</code>
Scientific Inferior: 12345	Scientific Inferior: 12345

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

`\aliasfontfeatureoption` If you wish to change the name of a font feature option, it can be aliased to another with the command `\aliasfontfeatureoption{}{<existing name>}{<new name>}`, such as shown in Example 57.

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

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

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

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 AAT feature (#1,#2).

<code>\fontspec_if_opentype:TF</code>	Test whether the currently selected font is an OpenType font. Always true for LuaTeX fonts.
<code>\fontspec_if_feature:nTF</code>	Test whether the currently selected font contains the raw OpenType feature #1. E.g.: <code>\fontspec_if_feature:nTF {pnum} {True} {False}</code> . Returns false if the font is not loaded by fontspec 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 fontspec 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 fontspec 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 fontspec 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 X_YTeX, because font slant information might be missing from the font. Therefore, we use L^AT_EX’s NFSS information, which is more likely to be correct.

20 Unicode footnote symbols

By default L^AT_EX 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, L^AT_EX 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 — 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_fontspeg_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 }
208 \msg_new:nnn {fontspec} {opa-twice}
209 {
210   Opacity~ set~ twice,~ in~ both~ Colour~ and~ Opacity.\
211   Using~ specification~ "Opacity=#1".
212 }
213 \msg_new:nnn {fontspec} {opa-twice-col}
214 {
215   Opacity~ set~ twice,~ in~ both~ Opacity~ and~ Colour.\
216   Using~ an~ opacity~ specification~ in~ hex~ of~ "#1/FF".

```

```

217 }
218 \msg_new:nnn {fontspec} {bad-colour}
219 {
220   Bad~ colour~ declaration~ "#1".~
221   Colour~ must~ be~ one~ of:\
222   *~ a~ named~ xcolor~ colour\
223   *~ a~ six-digit~ hex~ colour~ RRGGBB\
224   *~ an~ eight-digit~ hex~ colour~ RRGGBBTT~ with~ opacity
225 }

```

23.3 Option processing

```

226 \DeclareOption{cm-default}{
227   \fontspec_warning:n {cm-default-obsolete}
228 }
229 \DeclareOption{math}{\@zf@mathtrue}
230 \DeclareOption{no-math}{\@zf@mathfalse}
231 \DeclareOption{config}{\@zf@configfiletrue}
232 \DeclareOption{no-config}{\@zf@configfilefalse}
233 \DeclareOption{quiet}{
234   \msg_redirect_module:nnn { fontspec } { warning } { info }
235   \msg_redirect_module:nnn { fontspec } { info } { none }
236 }
237 \DeclareOption{silent}{
238   \msg_redirect_module:nnn { fontspec } { warning } { none }
239   \msg_redirect_module:nnn { fontspec } { info } { none }
240 }
241 \ExecuteOptions{config,math}
242 \ProcessOptions*

```

23.4 Packages

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

```

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

```

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

```

245 \RequirePackage{fontspec-patches}

```

23.5 Encodings

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

```

246 \xetex_or luatex:nnn {\tl_set:Nn \zf@enc} {EU1} {EU2}
247 \tl_set:Nn \rmdefault {lmr}
248 \tl_set:Nn \sfdefault {lmss}
249 \tl_set:Nn \ttdefault {lmtt}

```

```

250 \RequirePackage[\zf@enc]{fontenc}
251 \tl_set_eq:NN \UTFencname \zf@enc % for xunicode

```

Dealing with a couple of the problems introduced by babel:

```

252 \tl_set_eq:NN \cyrillicencoding \zf@enc
253 \tl_set_eq:NN \latinencoding \zf@enc
254 \g@addto@macro \document {
255   \tl_set_eq:NN \cyrillicencoding \zf@enc
256   \tl_set_eq:NN \latinencoding \zf@enc
257 }

```

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.

xunicode Now we load xunicode, working around its internal XeTeX check when under LuaTeX.

```

258 \xetex_or luatex:nn
259 {
260   \RequirePackage{xunicode}
261 }
262 {
263   \cs_set_eq:NN \fontspec_tmp: \XeTeXpicfile
264   \cs_set:Npn \XeTeXpicfile {}
265   \RequirePackage{xunicode}
266   \cs_set_eq:NN \XeTeXpicfile \fontspec_tmp:
267 }

```

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 23.9 on page 64](#).

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.

```

268 \DeclareDocumentCommand \fontspec { O{} m } {
269   \fontencoding {\zf@enc}
270   \fontspec_set_family:Nnn \f@family {#1}{#2}
271   \selectfont
272   \ignorespaces
273 }

```

`\setmainfont` The following three macros perform equivalent operations setting the default font (using `\let` rather than `\renewcommand` because `\zf@family` will change in the
`\setsansfont`
`\setmonofont`

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.

```
274 \DeclareDocumentCommand \setmainfont { O{} m } {
275   \fontspec_set_family:Nnn \rmdefault {#1}{#2}
276   \normalfont
277 }
278 \DeclareDocumentCommand \setsansfont { O{} m } {
279   \fontspec_set_family:Nnn \sfdefault {#1}{#2}
280   \normalfont
281 }
282 \DeclareDocumentCommand \setmonofont { O{} m } {
283   \fontspec_set_family:Nnn \ttdefault {#1}{#2}
284   \normalfont
285 }
```

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

```
286 \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 287 \DeclareDocumentCommand \setmathrm { O{} m } {
288   \fontspec_set_family:Nnn \zf@rmmaths {#1}{#2}
289 }
290 \DeclareDocumentCommand \setboldmathrm { O{} m } {
291   \fontspec_set_family:Nnn \zf@rmboldmaths {#1}{#2}
292 }
293 \DeclareDocumentCommand \setmathsf { O{} m } {
294   \fontspec_set_family:Nnn \zf@sfmaths {#1}{#2}
295 }
296 \DeclareDocumentCommand \setmathtt { O{} m } {
297   \fontspec_set_family:Nnn \zf@ttmaths {#1}{#2}
298 }
299 \@onlypreamble\setmathrm
300 \@onlypreamble\setboldmathrm
301 \@onlypreamble\setmathsf
302 \@onlypreamble\setmathtt
```

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

```
303 \def\zf@rmmaths{\rmdefault}
304 \def\zf@sfmaths{\sfdefault}
305 \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.

```

306 \DeclareDocumentCommand \newfontfamily { m O{} m } {
307   \fontspec_select:nn{#2}{#3}
308   \use:x {
309     \exp_not:N \DeclareRobustCommand \exp_not:N #1 {
310       \exp_not:N \fontencoding {\zf@enc}
311       \exp_not:N \fontfamily {\zf@family} \exp_not:N \selectfont
312     }
313   }
314 }

```

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

```

315 \DeclareDocumentCommand \newfontface { m O{} m } {
316   \newfontfamily #1 [ BoldFont={},ItalicFont={},SmallCapsFont={},#2 ] {#3}
317 }

```

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.

```

318 \DeclareDocumentCommand \defaultfontfeatures {m} {\def\zf@default@options{#1,}}
319 \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.

```

320 \DeclareDocumentCommand \addfontfeatures {m} {
321   \ifcsname zf@family@fontdef\zf@family\endcsname
322     \begingroup
323     \let\zf@default@options\@empty
324     \use:x {
325       \exp_not:N\fontspec_select:nn
326       {\csname zf@family@options\zf@family\endcsname,#1}
327       {\csname zf@family@fontname\zf@family\endcsname}
328     }

```

```

329     \endgroup
330     \fontfamily\zf@family\selectfont
331 \else
332     \fontspec_warning:n {addfontfeatures-ignored}
333 \fi
334 \ignorespaces
335 }
336 \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.

```

337 \DeclareDocumentCommand \newfontfeature {mm} {
338     \stepcounter{zf@newff}
339     \cs_set:cpx{+zf-#1}{+zf-\the\c@zf@newff}
340     \define@key{zf}[options]{#1}[] {
341         \zf@update@family{\csname+zf-#1\endcsname}
342         \zf@update@ff{#2}
343     }
344 }

```

`\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.

```

345 \DeclareDocumentCommand \newAATfeature {mmm} {
346     \unless\ifcsname zf@options@#1\endcsname
347         \zf@define@font@feature{#1}
348     \fi
349     \key@ifundefined{zf}{#1}{#2}{} {
350         \fontspec_warning:nxx {feature-option-overwrite}{#1}{#2}
351     }
352     \zf@define@feature@option{#1}{#2}{#3}{#4}{}
353 }

```

`\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.

```

354 \DeclareDocumentCommand \newICUfeature {mmm} {
355     \unless\ifcsname zf@options@#1\endcsname
356         \zf@define@font@feature{#1}
357     \fi
358     \key@ifundefined{zf}{#1}{#2}{} {
359         \fontspec_warning:nxx {feature-option-overwrite}{#1}{#2}
360     }
361     \zf@define@feature@option{#1}{#2}{}{}{#3}

```

```

362 }
363 \cs_set_eq:NN \newopentypefeature \newICUfeature

```

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

```

364 \DeclareDocumentCommand \aliasfontfeature {mm} {\multi@alias@key{#1}{#2}}
365 \DeclareDocumentCommand \aliasfontfeatureoption {mmm} {
366   \keyval@alias@key[zf@feat]{#1}{#2}{#3}
367 }

```

`\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.

```

368 \DeclareDocumentCommand \newfontscript {mm}
369 {
370   \fontspec_new_script:nn {#1} {#2}
371   \fontspec_new_script:nn {#2} {#2}
372 }
373 \cs_new:Npn \fontspec_new_script:nn #1#2
374 {
375   \define@key[zf@feat]{Script}{#1}[] {
376     \fontspec_check_script:nTF {#2} {
377       \zf@update@family{+script=#1}
378       \tl_set:Nn \l_fontspec_script_tl {#2}
379       \c@zf@script=\l_fontspec_strnum_int\relax
380     } {
381       \fontspec_warning:nx {script-not-exist} {#1}
382     }
383   }
384 }

```

`\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_YTeX) \font feature string with the appropriate language selection tag.

```

385 \DeclareDocumentCommand \newfontlanguage {mm}
386 {
387   \fontspec_new_lang:nn {#1} {#2}
388   \fontspec_new_lang:nn {#2} {#2}
389 }
390 \cs_new:Npn \fontspec_new_lang:nn #1#2
391 {
392   \define@key[zf@feat]{Lang}{#1}[] {
393     \fontspec_check_lang:nTF {#2} {
394       \zf@update@family{+lang=#1}
395       \tl_set:Nn \l_fontspec_lang_tl {#2}
396       \c@zf@language=\l_fontspec_strnum_int\relax
397     } {

```

```

398     \fontspec_warning:nx {language-not-exist} {#1}
399   }
400 }
401 }

```

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

```

402 \DeclareDocumentCommand \DeclareFontsExtensions {m}
403 {
404   \tl_set:Nx \l_fontspec_extensions_clist { \zap@space #1~\@empty }
405 }
406 \DeclareFontsExtensions{.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.

```

407 \prg_new_conditional:Nnn \fontspec_if_fontspec_font: {TF,T,F} {
408   \ifcsname zf@family@fontdef\f@family\endcsname
409     \prg_return_true:
410   \else
411     \prg_return_false:
412   \fi
413 }

```

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

```

414 \prg_new_conditional:Nnn \fontspec_if_aat_feature:nn {TF,T,F} {
415   \ifcsname zf@family@fontdef\f@family\endcsname
416     \font_set:Nnn \zf@basefont {\use:c{zf@family@fontdef\f@family}} {\f@size pt}
417     \ifzf@atsui
418       \fontspec_make_AAT_feature_string:nn{#1}{#2}
419       \ifx\@tempa\@empty
420         \prg_return_false:
421       \else
422         \prg_return_true:
423       \fi
424     \else
425       \prg_return_false:
426     \fi
427   \else
428     \prg_return_false:
429   \fi
430 }

```

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

```

431 \prg_new_conditional:Nnn \fontspec_if_opentype: {TF,T,F} {
432   \ifcsname zf@family@fontdef\f@family\endcsname
433   \font_set:Nnn \zf@basefont {\csname zf@family@fontdef\f@family\endcsname} {\f@size pt}
434   \fontspec_set_font_type:
435   \ifzf@icu
436     \prg_return_true:
437   \else
438     \prg_return_false:
439   \fi
440 \else
441   \prg_return_false:
442 \fi
443 }
```

`\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.

```

444 \prg_new_conditional:Nnn \fontspec_if_feature:n {TF,T,F} {
445   \ifcsname zf@family@fontdef\f@family\endcsname
446   \font_set:Nnn \zf@basefont {\csname zf@family@fontdef\f@family\endcsname} {\f@size pt}
447   \fontspec_set_font_type:
448   \ifzf@icu
449     \int_set:Nn \c@zf@script
450     {\use:c {g_fontspec_script_num_}(\zf@family)_tl}}
451     \int_set:Nn \c@zf@language
452     {\use:c {g_fontspec_lang_num_}(\zf@family)_tl}}
453     \tl_set:Nv \l_fontspec_script_tl {g_fontspec_script_}(\zf@family)_tl}
454     \tl_set:Nv \l_fontspec_lang_tl {g_fontspec_lang_}(\zf@family)_tl}
455     \fontspec_check_ot_feat:nTF {#1} \prg_return_true: \prg_return_false:
456   \else
457     \prg_return_false:
458   \fi
459 \else
460   \prg_return_false:
461 \fi
462 }
```

`\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.

```

463 \prg_new_conditional:Nnn \fontspec_if_feature:nnn {TF,T,F} {
464   \ifcsname zf@family@fontdef\f@family\endcsname
465   \font_set:Nnn \zf@basefont {\csname zf@family@fontdef\f@family\endcsname} {\f@size pt}
466   \fontspec_set_font_type:
467   \ifzf@icu
468     \fontspec_iv_str_to_num:n{#1} \c@zf@script = \l_fontspec_strnum_int \relax
469     \fontspec_iv_str_to_num:n{#2} \c@zf@language = \l_fontspec_strnum_int \relax
470     \fontspec_check_ot_feat:nTF {#3} \prg_return_true: \prg_return_false:
```

```

471 \else
472 \prg_return_false:
473 \fi
474 \else
475 \prg_return_false:
476 \fi
477 }

```

`\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.

```

478 \prg_new_conditional:Nnn \fontspec_if_script:n {TF,T,F} {
479 \ifcsname zf@family@fontdef\f@family\endcsname
480 \font_set:Nnn \zf@basefont {\csname zf@family@fontdef\f@family\endcsname} {\f@size pt}
481 \fontspec_set_font_type:
482 \ifzf@icu
483 \fontspec_check_script:nTF {#1} \prg_return_true: \prg_return_false:
484 \else
485 \prg_return_false:
486 \fi
487 \else
488 \prg_return_false:
489 \fi
490 }

```

`\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.

```

491 \prg_new_conditional:Nnn \fontspec_if_language:n {TF,T,F} {
492 \ifcsname zf@family@fontdef\f@family\endcsname
493 \font_set:Nnn \zf@basefont {\csname zf@family@fontdef\f@family\endcsname} {\f@size pt}
494 \fontspec_set_font_type:
495 \ifzf@icu
496 \tl_set:Nv \l_fontspec_script_tl {g_fontspec_script_(\zf@family)_tl}
497 \int_set:Nn \c_zf@script
498 {\use:c {g_fontspec_script_num_(\zf@family)_tl}}
499 \fontspec_check_lang:nTF {#1} \prg_return_true: \prg_return_false:
500 \else
501 \prg_return_false:
502 \fi
503 \else
504 \prg_return_false:
505 \fi
506 }

```

`\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.

```

507 \prg_new_conditional:Nnn \fontspec_if_language:nn {TF,T,F} {
508 \ifcsname zf@family@fontdef\f@family\endcsname
509 \font_set:Nnn \zf@basefont {\csname zf@family@fontdef\f@family\endcsname} {\f@size pt}

```

```

510 \fontspec_set_font_type:
511 \ifzf@icu
512 \tl_set:Nn \l_fontspec_script_tl {#1}
513 \fontspec_iv_str_to_num:n{#1} \c@zf@script = \l_fontspec_strnum_int \relax
514 \fontspec_check_lang:nTF {#2} \prg_return_true: \prg_return_false:
515 \else
516 \prg_return_false:
517 \fi
518 \else
519 \prg_return_false:
520 \fi
521 }

```

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

```

522 \prg_new_conditional:Nnn \fontspec_if_current_script:n {TF,T,F} {
523 \ifcsname zf@family@fontdef\f@family\endcsname
524 \font_set:Nnn \zf@basefont {\csname zf@family@fontdef\f@family\endcsname} {\f@size pt}
525 \fontspec_set_font_type:
526 \ifzf@icu
527 \tl_if_eq:nvTF {#1} {g_fontspec_script_(\zf@family)_tl}
528 {\prg_return_true:} {\prg_return_false:}
529 \else
530 \prg_return_false:
531 \fi
532 \else
533 \prg_return_false:
534 \fi
535 }

```

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

```

536 \prg_new_conditional:Nnn \fontspec_if_current_language:n {TF,T,F} {
537 \ifcsname zf@family@fontdef\f@family\endcsname
538 \font_set:Nnn \zf@basefont {\csname zf@family@fontdef\f@family\endcsname} {\f@size pt}
539 \fontspec_set_font_type:
540 \ifzf@icu
541 \tl_if_eq:nvTF {#1} {g_fontspec_lang_(\zf@family)_tl}
542 {\prg_return_true:} {\prg_return_false:}
543 \else
544 \prg_return_false:
545 \fi
546 \else
547 \prg_return_false:
548 \fi
549 }

```

Need this:

```
550 \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 $\langle features \rangle$ and $\langle font \rangle$, and stores the name in the variable $\langle family \rangle$. See the standard fontspec user commands for applications of this function.

We want to store the actual name of the font family within the $\langle family \rangle$ 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.

```
551 \cs_new:Npn \fontspec_set_family:Nnn #1#2#3 {
552   \fontspec_select:nn {#2}{#3}
553   \tl_set_eq:NN #1 \zf@family
554 }
```

23.8 expl3 interface for font loading

Using `\font\1={xyz}` notation doesn't work at present with LuaLaTeX, although it should be the correct way to do things. Continue to use `\font\1="xyz"` for now for both engines.

```
555 \xetex_or luatex:nnn { \cs_set:Npn \fontspec_fontwrap:n #1 } { "#1" } { {#1} }
556 \cs_set:Npn \fontspec_fontwrap:n #1 { "#1" }
```

Beginnings of an 'l3font', I guess:

```
557 \cs_new:Npn \font_set:Nnn #1#2#3 {
558   \font #1 = \fontspec_fontwrap:n {#2} ~at~#3\scan_stop:
559 }
560 \cs_new:Npn \font_gset:Nnn #1#2#3 {
561   \global \font #1 = \fontspec_fontwrap:n {#2} ~at~#3\scan_stop:
562 }
```

23.9 Internal macros

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

`\fontspec_select:nn` This is the command that defines font families for use, the underlying procedure of all `\fontspec`-like commands. Given a list of font features (#1) for a requested font (#2), it will define an NFSS family for that font and put the family name (globally) into `\zf@family`. The L^AT_EX '`\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.

```
563 \cs_set:Npn \fontspec_select:nn #1#2 {
564   \begingroup
565   \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.


```

566 \edef\zf@fontname{#2}
567 \let\zf@family@long\zf@fontname
568 \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.

```

569 \fontspec_preparse_features:nn {#1}{#2}
570 \fontspec_set_scriptlang:
571 \fontspec_get_features:n {\zf@font@feat}
572 \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`.

```

573 \fontspec_save_family:nT {#2} {
574   \fontspec_info:nxx {defining-font} {#1} {#2}
575   \fontspec_save_fontinfo:nn {#1} {#2}
576   \DeclareFontFamily{\zf@enc}{\zf@family}{}
577   \fontspec_set_upright:
578   \fontspec_set_bold:
579   \fontspec_set_italic:
580   \fontspec_set_slanted:
581   \fontspec_set_bold_italic:
582   \fontspec_set_bold_slanted:
583 }
584 \endgroup
585 }

```

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

```

586 \cs_set_eq:NN \zf@fontspec \fontspec_select:nn

```

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

```

587 \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.

```

588 \fontspec_if_detect_external:nT {#2}
589 { \setkeys[zf]{preparse-external}{ExternalLocation} }
590 \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.

```

591 \let\zf@fontname\zf@up
592 \fontspec_setkeys:xx {preparse} {\XKV@rm}
593 \let\zf@font@feat\XKV@rm

```

Finally save the ‘confirmed’ font definition.

```

594 \font_set:Nnn \zf@basefont {\fontspec_fullname:n {\zf@up}} {\f@size pt}
595 \fontspec_set_font_type:
596 \font_gset:Nnn \zf@basefont {\fontspec_fullname:n {\zf@up}} {\f@size pt}
597 \zf@basefont % this is necessary for LuaLaTeX to check the scripts properly
598 }

```

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

```

599 \prg_new_conditional:Nnn \fontspec_if_detect_external:n {T}
600 {
601   \clist_map_inline:Nn \l_fontspec_extensions_clist
602   {
603     \bool_set_false:N \l_tmpa_bool
604     \tl_if_in:nnT {#1 = end_of_string} {##1 = end_of_string}
605     { \bool_set_true:N \l_tmpa_bool \clist_map_break: }
606   }
607   \bool_if:NTF \l_tmpa_bool \prg_return_true: \prg_return_false:
608 }

```

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

```

609 \cs_set:Npn \fontspec_fullname:n #1 {
610   \fontspec_namewrap:n { #1 \l_fontspec_extension_tl }
611   \l_fontspec_renderer_tl
612   \l_fontspec_optical_size_tl
613 }

```

`\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.

```

614 \prg_new_conditional:Nnn \fontspec_save_family:n {T} {
615   \cs_if_exist:cF {g_fontspec_UID_\zf@family@long}
616   {
617     \cs_if_exist:cTF {g_fontspec_family_#1_int} {
618       \int_gincr:c {g_fontspec_family_#1_int}
619     }{
620       \int_new:c {g_fontspec_family_#1_int}
621     }
622     \edef\@tempa{#1~}
623     \tl_gset:cx {g_fontspec_UID_\zf@family@long} {
624       \expandafter\zap@space\@tempa\@empty
625       ( \int_use:c {g_fontspec_family_#1_int} )
626     }
627   }
628   \tl_gset:Nv \zf@family {g_fontspec_UID_\zf@family@long}
629   \cs_if_exist:cTF {zf@family@fontname\zf@family}

```

```

630 \prg_return_false: \prg_return_true:
631 }

```

`\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.

```

632 \cs_new:Npn \fontspec_set_scriptlang: {
633   \ifzf@icu
634     \tl_if_empty:NTF \l_fontspec_script_name_tl {
635       \fontspec_check_script:nTF {latn}
636       {
637         \tl_set:Nn \l_fontspec_script_name_tl {Latin}
638         \tl_if_empty:NT \l_fontspec_lang_name_tl {
639           \tl_set:Nn \l_fontspec_lang_name_tl {Default}
640         }
641         \fontspec_setkeys:xxx {feat} {Script} {\l_fontspec_script_name_tl}
642         \fontspec_setkeys:xxx {feat} {Lang}   {\l_fontspec_lang_name_tl}
643       }
644     {
645       \fontspec_info:n {no-scripts}
646     }
647   }
648   {
649     \tl_if_empty:NT \l_fontspec_lang_name_tl {
650       \tl_set:Nn \l_fontspec_lang_name_tl {Default}
651     }
652     \fontspec_setkeys:xxx {feat} {Script} {\l_fontspec_script_name_tl}
653     \fontspec_setkeys:xxx {feat} {Lang}   {\l_fontspec_lang_name_tl}
654   }
655   \fi
656 }

```

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

```

657 \cs_new:Npn \fontspec_save_fontinfo:nn #1#2 {
658   \tl_gset:cx {zf@family@fontname\zf@family} {#2}
659   \tl_gset:cx {zf@family@options\zf@family} {\zf@default@options #1}
660   \tl_gset:cx {zf@family@fontdef\zf@family} {
661     \fontspec_fullname:n {\zf@fontname} : \l_fontspec_pre_feat_tl \l_fontspec_rawfeatures_sclist
662   }
663   \tl_gset:cx {g_fontspec_script_num_(\zf@family)_tl}
664     { \int_use:N \c@zf@script   }
665   \tl_gset:cx {g_fontspec_lang_num_(\zf@family)_tl}
666     { \int_use:N \c@zf@language }
667   \tl_gset:eq:cN {g_fontspec_script_(\zf@family)_tl} \l_fontspec_script_tl
668   \tl_gset:eq:cN {g_fontspec_lang_(\zf@family)_tl}   \l_fontspec_lang_tl
669 }

```

`\fontspec_set_upright:` Sets the upright shape.

```

670 \cs_new:Npn \fontspec_set_upright: {
671   \zf@make@font@shapes{\zf@fontname}
672   {\mddefault}{\updefault}{\zf@font@feat\zf@up@feat}
673 }

```

`\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.

```
674 \cs_new:Npn \fontspec_set_bold: {  
675   \unless\ifzf@nobf  
676     \ifx\zf@bf\empty  
677       \zf@make@font@shapes[\zf@fontname]{/B}  
678       {\bfdefault}{\updefault}{\zf@font@feat\zf@bf@feat}  
679     \else  
680       \zf@make@font@shapes{\zf@bf}  
681       {\bfdefault}{\updefault}{\zf@font@feat\zf@bf@feat}  
682     \fi  
683   \fi  
684 }
```

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

```
685 \cs_new:Npn \fontspec_set_italic: {  
686   \unless\ifzf@noit  
687     \ifx\zf@it\empty  
688       \zf@make@font@shapes[\zf@fontname]{/I}  
689       {\mddefault}{\itdefault}{\zf@font@feat\zf@it@feat}  
690     \else  
691       \zf@make@font@shapes{\zf@it}  
692       {\mddefault}{\itdefault}{\zf@font@feat\zf@it@feat}  
693     \fi  
694   \fi  
695 }
```

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

```
696 \cs_new:Npn \fontspec_set_slanted: {  
697   \ifx\zf@sl\empty\else  
698     \zf@make@font@shapes{\zf@sl}  
699     {\mddefault}{\sldefault}{\zf@font@feat\zf@sl@feat}  
700   \fi  
701 }
```

`\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.

```
702 \cs_new:Npn \fontspec_set_bold_italic: {  
703   \@tempswatrue  
704   \ifzf@nobf\@tempswafalse\fi  
705   \ifzf@noit\@tempswafalse\fi  
706   \bool_if:NT \l_fontspec_external_bool \@tempswafalse  
707   \if@tempswa
```

```

708 \ifx\zf@bfit\empty
709 \ifx\zf@bf\empty
710 \ifx\zf@it\empty
711 \zf@make@font@shapes[\zf@fontname]{/BI}
712 {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}
713 \else
714 \zf@make@font@shapes[\zf@it]{/B}
715 {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}
716 \fi
717 \else
718 \zf@make@font@shapes[\zf@bf]{/I}
719 {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}
720 \fi
721 \else
722 \zf@make@font@shapes{\zf@bfit}
723 {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}
724 \fi
725 \fi
726 }

```

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

```

727 \cs_new:Npn \fontspec_set_bold_slanted: {
728 \ifx\zf@bfs\empty
729 \ifx\zf@sl\empty\else
730 \zf@make@font@shapes[\zf@sl]{/B}
731 {\bfdefault}{\sldefault}{\zf@font@feat\zf@bfs\@feat}
732 \fi
733 \else
734 \zf@make@font@shapes{\zf@bfs\}
735 {\bfdefault}{\sldefault}{\zf@font@feat\zf@bfs\@feat}
736 \fi
737 }

```

23.9.1 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.

```

738 \xetex_or luatex:nnn { \cs_new:Npn \fontspec_set_font_type: }
739 {
740 \zf@tfmfalse \zf@atsuifalse \zf@icufalse \zf@mmfalse \zf@graphitefalse
741 \ifcase\XeTeXfonttype\zf@basefont
742 \zf@tfmtrue
743 \or
744 \zf@atsui true
745 \ifnum\XeTeXcountvariations\zf@basefont < \c_zero
746 \zf@mmtrue
747 \fi
748 \or

```

```

749     \zf@icutrue
750     \fi

```

If automatic, the `\l_fontspec_renderer_tl` token list will still be empty (other suffices that could be added will be later in the feature processing), and if it is indeed still empty, assign it a value so that the other weights of the font are specifically loaded with the same renderer.

```

751     \tl_if_empty:NT \l_fontspec_renderer_tl {
752         \ifzf@atsui
753             \tl_set:Nn \l_fontspec_renderer_tl {/AAT}
754         \else\ifzf@icu
755             \tl_set:Nn \l_fontspec_renderer_tl {/ICU}
756         \fi\fi
757     }
758 }
759 {
760     \zf@icutrue
761 }

```

`\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.

```

762 \newcommand*\zf@make@font@shapes[5][]{
763     \begingroup
764     \edef\@tempa{#1}
765     \unless\ifx\@tempa\@empty
766         \font_set:Nnn \@tempfonta {\fontspec_fullname:n {#1}} {\f@size pt}
767         \edef\@tempa{\fontname\@tempfonta}
768     \fi
769     \font_set:Nnn \@tempfontb {\fontspec_fullname:n {#1#2}} {\f@size pt}
770     \edef\@tempb{\fontname\@tempfontb}
771     \ifx\@tempa\@tempb
772         \fontspec_info:nx {no-font-shape} {#1#2}
773     \else
774         \edef\zf@fontname{#1#2}
775         \let\zf@basefont\@tempfontb
776         \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 23.11 on page 99](#) for the code that enables this usage.

```

777 \ifx\zf@sc\@empty
778 \unless\ifzf@nosc
779 \zf@make@smallcaps
780 \unless\ifx\zf@smallcaps\@empty
781 \fontspec_declare_shape:nnnn {\zf@smallcaps}{#3}
782 {\ifx#4\itdefault\sidefault\else\scdefault\fi}{#5\zf@sc@feat}
783 \fi
784 \fi
785 \else
786 \edef\zf@fontname{\zf@sc}
787 \fontspec_declare_shape:nnnn {}{#3}
788 {\ifx#4\itdefault\sidefault\else\scdefault\fi}{#5\zf@sc@feat}
789 \fi
790 \fi
791 \endgroup
792 }

```

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`.

```

793 \cs_new:Npn \fontspec_declare_shape:nnnn #1#2#3#4 {
794 \clist_if_empty:NTF \l_fontspec_sizefeat_clist
795 {
796 \fontspec_get_features:n{#4}
797 \tl_set:Nx \l_fontspec_nfss_tl {
798 - \l_fontspec_scale_tl
799 \fontspec_fontwrap:n {
800 \fontspec_fullname:n {\zf@fontname} :
801 \l_fontspec_pre_feat_tl \l_fontspec_rawfeatures_sclist #1
802 }
803 }
804 }

```

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}}.`

```

805 {
806 \tl_clear:N \l_fontspec_nfss_tl
807 \clist_map_inline:Nn \l_fontspec_sizefeat_clist {
808 \tl_clear:N \l_fontspec_size_tl
809 \tl_set_eq:NN \l_fontspec_sizedfont_tl \zf@fontname
810 \fontspec_setkeys:xx {sizing} { \expandafter \@firstofone ##1 }
811 \tl_if_empty:NT \l_fontspec_size_tl { \fontspec_error:n {no-size-info} }
812 \fontspec_get_features:n{#4,\XKV@rm}

```

```

813 \tl_put_right:Nx \l_fontspec_nfss_tl {
814   \l_fontspec_size_tl \l_fontspec_scale_tl
815   \fontspec_fontwrap:n {
816     \fontspec_fullname:n { \l_fontspec_sizedfont_tl }
817     : \l_fontspec_pre_feat_tl \l_fontspec_rawfeatures_sclist #1
818   }
819 }
820 }
821 }

```

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.

```

822 \PackageInfo{fontspec}{
823   Defining~ shape~ #2/#3~ with~ raw~ font~ features:
824   \MessageBreak \l_fontspec_rawfeatures_sclist
825   \@gobble}
826 \use:x{
827   \exp_not:N\DeclareFontShape{\zf@enc}{\zf@family}{#2}{#3}
828     {\l_fontspec_nfss_tl}{\zf@adjust}
829 }

```

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?

```

830 \tl_if_eq:xxT {#3} {\itdefault}
831 {
832   \use:x {
833     \exp_not:N \DeclareFontShape {\zf@enc}{\zf@family}{#2}{\sldefault}
834       {-ssub*\zf@family/#2/\itdefault}{\zf@adjust}
835   }
836 }
837 }

```

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

```

838 \xetex_or luatex:nnn { \tl_set:Nn \l_fontspec_pre_feat_tl }
839 {
840   \ifzf@icu
841     \tl_if_empty:NF \l_fontspec_script_tl
842     {
843       script = \l_fontspec_script_tl ;
844       language = \l_fontspec_lang_tl ;
845     }
846   \fi
847 }
848 {
849   mode = \l_fontspec_mode_tl ;
850   \tl_if_empty:NF \l_fontspec_script_tl
851   {
852     script = \l_fontspec_script_tl ;
853     language = \l_fontspec_lang_tl ;
854   }

```



```
855 }
```

`\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.

```
856 \newcommand*{\zf@update@family}[1]{
857   \ifzf@firsttime
858     \xdef\zf@family@long{\zf@family@long#1}
859   \fi
860 }
```

23.9.2 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.

```
861 \cs_set:Npn \fontspec_get_features:n #1 {
862   \let\l_fontspec_rawfeatures_sclist \empty
863   \tl_clear:N \l_fontspec_scale_tl
864   \tl_set_eq:NN \l_fontspec_opacity_tl \g_fontspec_opacity_tl
865   \tl_set_eq:NN \l_fontspec_hexcol_tl \g_fontspec_hexcol_tl
866   \let\zf@adjust \empty
867   \fontspec_setkeys:xx {options} {#1}
868   \tl_if_empty:NF \XKV@rm {
869     \fontspec_error:nx {unknown-options} { \exp_not:V \XKV@rm }
870   }
```

Finish the colour specification:

```
871   \zf@update@ff{color=\l_fontspec_hexcol_tl\l_fontspec_opacity_tl}
872 }
```

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

```
873 \tl_clear:N \zf@bf
874 \tl_clear:N \zf@it
875 \tl_clear:N \zf@fake@slant
876 \tl_clear:N \zf@fake@embolden
877 \tl_clear:N \zf@bfit
878 \tl_clear:N \zf@sl
879 \tl_clear:N \zf@bfsl
880 \tl_clear:N \zf@sc
881 \tl_clear:N \zf@up@feat
882 \tl_clear:N \zf@bf@feat
883 \tl_clear:N \zf@it@feat
884 \tl_clear:N \zf@bfit@feat
885 \tl_clear:N \zf@sl@feat
886 \tl_clear:N \zf@bfsl@feat
887 \tl_clear:N \zf@sc@feat
888 \tl_clear:N \l_fontspec_script_name_tl
889 \tl_clear:N \l_fontspec_script_tl
890 \tl_clear:N \l_fontspec_lang_name_tl
```

```

891 \tl_clear:N \l_fontspec_lang_tl
892 \clist_clear:N \l_fontspec_sizefeat_clist
893 \tl_new:Nn \g_fontspec_hexcol_tl {000000}
894 \tl_new:Nn \g_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.)

```

895 \newcommand*\fontspec_init:{
896   \zf@icufalse
897   \zf@firsttimetrue
898   \xetex_or luatex:nnn { \cs_set:Npn \fontspec_namewrap:n ##1 }
899     { ##1 }
900     { name:##1 }
901   \tl_clear:N \l_fontspec_optical_size_tl
902   \tl_clear:N \l_fontspec_renderer_tl
903   \luatex_if_engine:T {
904     \tl_set:Nn \l_fontspec_mode_tl {node}
905     \luatexprehyphenchar = '\- % fixme
906     \luatexposthyphenchar = 0 % fixme
907     \luatexpreexhyphenchar = 0 % fixme
908     \luatexpostexhyphenchar = 0 % fixme
909   }
910 }

```

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

```

911 \newcommand*\zf@make@smallcaps{
912   \let\zf@smallcaps\@empty
913   \xetex_or luatex:nn
914   {
915     \ifzf@atsui
916       \fontspec_make_AAT_feature_string:nn{3}{3}
917       \unless\ifx\@tempa\@empty
918         \edef\zf@smallcaps{\@tempa;}
919       \fi
920     \fi
921     \ifzf@icu
922       \fontspec_check_ot_feat:nT {+smcp} {\edef\zf@smallcaps{+smcp;}}
923     \fi
924   }
925   {
926     \fontspec_check_ot_feat:nT {+smcp} {\edef\zf@smallcaps{+smcp;}}
927   }
928 }

```

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

```

929 \cs_new:Npn \sclist_put_right:Nn #1#2 {
930   \tl_if_empty:NTF #1 {
931     \tl_set:Nn #1 {#2}
932   }{
933     \tl_put_right:Nn #1 {;#2}
934   }

```

935 }

\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.

```
936 \newcommand*\zf@update@ff[1]{
937   \unless\ifzf@firsttime
938     \xdef\l_fontspec_rawfeatures_sclist{\l_fontspec_rawfeatures_sclist #1;}
939   \fi
940 }
```

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

```
941 \cs_new:Npn \fontspec_make_feature:nnn #1#2#3 {
942   \xetex_or luatex:nn
943   {
944     \ifzf@atsui
945       \fontspec_make_AAT_feature:nn {#1}{#2}
946     \fi
947     \ifzf@icu
948       \fontspec_make_ICU_feature:n {#3}
949     \fi
950   }
951   {
952     \fontspec_make_ICU_feature:n {#3}
953   }
954 }
955 \cs_generate_variant:Nn \fontspec_make_feature:nnn {nnx}

956 \cs_new:Npn \fontspec_make_AAT_feature:nn #1#2 {
957   \tl_if_empty:nTF {#1}
958   {
959     \fontspec_warning:n {aat-feature-not-exist}
960   }
961   {
962     \fontspec_make_AAT_feature_string:nn {#1}{#2}
963     \ifx\@tempa\@empty
964       \fontspec_warning:nx {aat-feature-not-exist-in-font} {#1,#2}
965     \else
966       \zf@update@family{+#1,#2}
967       \zf@update@ff\@tempa
968     \fi
969   }
970 }
971 \cs_new:Npn \fontspec_make_ICU_feature:n #1 {
972   \tl_if_empty:nTF {#1}
973   {
974     \fontspec_warning:n {icu-feature-not-exist}
975   }
976   {
977     \fontspec_check_ot_feat:nTF {#1} {
978       \zf@update@family{#1}
```

```

979     \zf@update@ff{#1}
980   }{
981     \fontspec_warning:nx {icu-feature-not-exist-in-font} {#1}
982   }
983 }
984 }

```

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

```

\zf@define@feature@option 985 \newcommand*\zf@define@font@feature[1]{
986   \define@key[zf]{options}{#1}{\setkeys[zf@feat]{#1}{##1}}
987 }
988 \newcommand*\zf@define@feature@option[5]{
989   \define@key[zf@feat]{#1}{#2}[\fontspec_make_feature:nnn{#3}{#4}{#5}]
990 }

```

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

```

991 \newcommand*\keyval@alias@key[4][KV]{
992   \cs_set_eq:cc{#1@#2@#4}{#1@#2@#3}
993   \cs_set_eq:cc{#1@#2@#4@default}{#1@#2@#3@default}
994 }

```

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

```

995 \newcommand*\multi@alias@key[2]{
996   \key@ifundefined[zf]{options}{#1}
997   {
998     \key@ifundefined[zf]{preparse}{#1}
999     {
1000       \key@ifundefined[zf]{preparse-external}{#1}
1001       { \fontspec_warning:nx {rename-feature-not-exist} {#1} }
1002       { \keyval@alias@key[zf]{preparse-external}{#1}{#2} }
1003     }
1004     { \keyval@alias@key[zf]{preparse}{#1}{#2} }
1005   }
1006   { \keyval@alias@key[zf]{options}{#1}{#2} }
1007 }

```

`\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 74).

```

1008 \cs_new:Npn \fontspec_make_AAT_feature_string:nn #1#2 {
1009   \edef\@tempa{\XeTeXfeaturename\zf@basefont #1}
1010   \unless\ifx\@tempa\@empty

```

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

```

1011   \ifnum\XeTeXisexclusivefeature\zf@basefont #10
1012   \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:

```

1013 \else
1014 \unless\ifodd #2
1015 \edef\@tempb{\XeTeXselectorname\zf@basefont #1\space #2}

```

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

```

1016 \else
1017 \edef\@tempb{
1018 \XeTeXselectorname\zf@basefont #1\space \numexpr#2-1\relax
1019 }
1020 \unless\ifx\@tempb\@empty
1021 \edef\@tempb{!\@tempb}
1022 \fi
1023 \fi
1024 \fi

```

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

```

1025 \unless\ifx\@tempb\@empty
1026 \edef\@tempa{\@tempa=\@tempb}
1027 \else
1028 \let\@tempa\@empty
1029 \fi
1030 \fi
1031 }

```

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

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

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

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

```

1032 \cs_set:Npn \fontspec_iv_str_to_num:n #1 {
1033 \fontspec_iv_str_to_num:w #1 \@empty \@empty \q_nil
1034 }
1035 \cs_set:Npn \fontspec_iv_str_to_num:w #1#2#3#4#5 \q_nil {
1036 \int_set:Nn \l_fontspec_strnum_int {
1037 '#1 * "1000000
1038 + '#2 * "10000
1039 + \ifx \@empty #3 32 \else '#3 \fi * "100

```

```

1040 + \ifx \@empty #4 32 \else ‘#4 \fi
1041 }
1042 }
1043 \cs_set:Npn \fontspec_v_str_to_num:n #1 {
1044   \bool_if:nTF
1045     {
1046       \tl_if_head_eq_charcode_p:nN {#1} {+} ||
1047       \tl_if_head_eq_charcode_p:nN {#1} {-}
1048     }
1049     {
1050       \exp_after:wN \fontspec_iv_str_to_num:n
1051       \exp_after:wN { \use_none:n #1 }
1052     }
1053     { \fontspec_iv_str_to_num:n {#1} }
1054 }

```

`\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.

```

1055 \xetex_or luatex:nnn {\prg_new_conditional:Nnn \fontspec_check_script:n {TF}}
1056 {
1057   \fontspec_iv_str_to_num:n{#1}
1058   \@tempcntb\XeTeXOTcountscripts\zf@basefont
1059   \c@zf@index\z@ \@tempwafalse
1060   \loop\ifnum\c@zf@index\@tempcntb
1061     \ifnum\XeTeXOTscripttag\zf@basefont\c@zf@index=\l_fontspec_strnum_int
1062       \@tempwattrue
1063       \c@zf@index\@tempcntb
1064     \else
1065       \advance\c@zf@index\@ne
1066     \fi
1067   \repeat
1068   \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
1069 }
1070 {
1071   \directlua{fontspec.check_ot_script("zf@basefont", "#1")}
1072   \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
1073 }

```

`\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’.

```

1074 \xetex_or luatex:nnn {\prg_new_conditional:Nnn \fontspec_check_lang:n {TF}}
1075 {
1076   \fontspec_iv_str_to_num:n{#1}
1077   \@tempcntb\XeTeXOTcountlanguages\zf@basefont\c@zf@script
1078   \c@zf@index\z@
1079   \@tempwafalse
1080   \loop\ifnum\c@zf@index\@tempcntb

```

```

1081     \ifnum\XeTeXOTlanguetag\zf@basefont\c@zf@script\c@zf@index=\l_fontspec_strnum_int
1082     \@tempswatrue
1083     \c@zf@index\@tempcntb
1084     \else
1085     \advance\c@zf@index\@ne
1086     \fi
1087     \repeat
1088     \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
1089 }
1090 {
1091     \directlua{
1092         fontspec.check_ot_lang( "zf@basefont", "#1", "\l_fontspec_script_tl" )
1093     }
1094     \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
1095 }

```

`\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'.

```

1096 \xetex_or luatex:nnn
1097 { \prg_new_conditional:Nnn \fontspec_check_ot_feat:n {TF,T} }
1098 {
1099     \@tempcntb\XeTeXOTcountfeatures\zf@basefont\c@zf@script\c@zf@language
1100     \fontspec_v_str_to_num:n {#1}
1101     \c@zf@index\z@
1102     \@tempwafalse
1103     \loop\ifnum\c@zf@index\@tempcntb
1104         \ifnum\XeTeXOTfeaturetag\zf@basefont\c@zf@script\c@zf@language
1105             \c@zf@index=\l_fontspec_strnum_int
1106             \@tempswatrue
1107             \c@zf@index\@tempcntb
1108         \else
1109             \advance\c@zf@index\@ne
1110         \fi
1111     \repeat
1112     \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
1113 }
1114 {
1115     \directlua{
1116         fontspec.check_ot_feat(
1117             "zf@basefont", "#1",
1118             "\l_fontspec_lang_tl", "\l_fontspec_script_tl"
1119         )
1120     }
1121     \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
1122 }

```

23.10 keyval definitions

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

23.10.1 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.

```
1123 \bool_new:N \l_fontspec_external_bool
1124 \define@key[zf]{preparse-external}{ExternalLocation}[]{
1125   \zf@nobftrue
1126   \zf@noittrue
1127   \bool_set_true:N \l_fontspec_external_bool
1128   \xetex_or luatex:nnn { \cs_gset:Npn \fontspec_namewrap:n ##1 }
1129   { [ #1 ##1 ] }
1130   { file: #1 ##1 }
1131   \xetex_if_engine:T { \setkeys[zf]{preparse}{Renderer=ICU} }
1132 }
1133 \aliasfontfeature{ExternalLocation}{Path}
```

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

```
1134 \define@key[zf]{preparse-external}{Extension}{
1135   \tl_set:Nn \l_fontspec_extension_tl {#1}
1136   \bool_if:NF \l_fontspec_external_bool {
1137     \setkeys*[zf]{preparse-external}{ExternalLocation}
1138   }
1139 }
1140 \tl_clear:N \l_fontspec_extension_tl
```

23.10.2 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.

```
1141 \define@choicekey[zf]{preparse}{Renderer}[\l_tmpa_tl\l_tmpa_num]
1142   {AAT,ICU,Graphite,Full,Basic}{
1143   \zf@update@family{+rend:#1}
1144   \intexpr_compare:nTF {\l_tmpa_num 3} {
1145     \xetex_or luatex:nn
```



```

1146 {
1147   \tl_set:Nv \l_fontspec_renderer_tl {g_fontspec_renderer_tag_\l_tmpa_tl}
1148 }
1149 {
1150   \fontspec_warning:nx {only-xetex-feature} {Renderer=AAT/ICU/Graphite}
1151 }
1152 }{
1153   \xetex_or luatex:nn
1154   { \fontspec_warning:nx {only-luatex-feature} {Renderer=Full/Basic} }
1155   { \tl_set:Nv \l_fontspec_mode_tl {g_fontspec_mode_tag_\l_tmpa_tl} }
1156 }
1157 }
1158 \tl_set:cn {g_fontspec_renderer_tag_AAT} {/AAT}
1159 \tl_set:cn {g_fontspec_renderer_tag_ICU} {/ICU}
1160 \tl_set:cn {g_fontspec_renderer_tag_Graphite} {/GR}
1161 \tl_set:cn {g_fontspec_mode_tag_Full} {node}
1162 \tl_set:cn {g_fontspec_mode_tag_Basic} {base}

```

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

```

1163 \define@key[zf]{prepare}{Script}{
1164   \xetex_if_engine:T { \setkeys[zf]{prepare}{Renderer=ICU} }
1165   \tl_set:Nn \l_fontspec_script_name_tl {#1}
1166   \zf@update@family{+script:#1}
1167 }

```

Exactly the same:

```

1168 \define@key[zf]{prepare}{Language}{
1169   \xetex_if_engine:T { \setkeys[zf]{prepare}{Renderer=ICU} }
1170   \tl_set:Nn \l_fontspec_lang_name_tl {#1}
1171   \zf@update@family{+language:#1}
1172 }

```

23.10.3 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:

```

1173 \define@key[zf]{prepare-external}{UprightFont}{
1174   \fontspec_complete_fontname:Nn \zf@up {#1}
1175   \zf@update@family{up:#1}
1176 }

```

Bold:

```

1177 \define@key[zf]{prepare-external}{BoldFont}{
1178   \edef\@tempa{#1}
1179   \ifx\@tempa\empty
1180     \zf@nobftrue
1181     \zf@update@family{nobf}
1182   \else

```

```

1183 \zf@nobffalse
1184 \fontspec_complete_fontname:Nn \zf@bf {#1}
1185 \zf@update@family{bf:#1}
1186 \fi
1187 }

```

Same for italic:

```

1188 \define@key[zf]{preparse-external}{ItalicFont}{
1189 \edef\@tempa{#1}
1190 \ifx\@tempa\@empty
1191 \zf@noittrue
1192 \zf@update@family{noit}
1193 \else
1194 \zf@noitfalse
1195 \fontspec_complete_fontname:Nn \zf@it {#1}
1196 \zf@update@family{it:#1}
1197 \fi
1198 }

```

Simpler for bold+italic & slanted:

```

1199 \define@key[zf]{preparse-external}{BoldItalicFont}{
1200 \fontspec_complete_fontname:Nn \zf@bfit {#1}
1201 \zf@update@family{bfit:#1}
1202 }
1203 \define@key[zf]{preparse-external}{SlantedFont}{
1204 \fontspec_complete_fontname:Nn \zf@sl {#1}
1205 \zf@update@family{sl:#1}
1206 }
1207 \define@key[zf]{preparse-external}{BoldSlantedFont}{
1208 \fontspec_complete_fontname:Nn \zf@bfs1 {#1}
1209 \zf@update@family{bfs1:#1}
1210 }

```

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

```

1211 \define@key[zf]{options}{SmallCapsFont}{
1212 \edef\@tempa{#1}
1213 \ifx\@tempa\@empty
1214 \zf@nosctrue
1215 \zf@update@family{nosc}
1216 \else
1217 \zf@noscfalse
1218 \fontspec_complete_fontname:Nn \zf@sc {#1}
1219 \zf@update@family{sc:\zap@space #1~\@empty}
1220 \fi
1221 }

```

`\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").

```

1222 \cs_set:Npn \fontspec_complete_fontname:Nn #1#2 {
1223 \tl_set:Nn #1 {#2}
1224 \tl_replace_all_in:Nnx #1 {*} {\zf@fontname}
1225 }

```

```
1226 \cs_generate_variant:Nn \tl_replace_all_in:Nnn {Nnx}
```

Features

```
1227 \define@key[zf]{preparse}{UprightFeatures}{
1228   \def\zf@up@feat{, #1}
1229   \zf@update@family{rmfeat:#1}
1230 }
1231 \define@key[zf]{preparse}{BoldFeatures}{
1232   \def\zf@bf@feat{, #1}
1233   \zf@update@family{bf feat:#1}
1234 }
1235 \define@key[zf]{preparse}{ItalicFeatures}{
1236   \def\zf@it@feat{, #1}
1237   \zf@update@family{it feat:#1}
1238 }
1239 \define@key[zf]{preparse}{BoldItalicFeatures}{
1240   \def\zf@bfit@feat{, #1}
1241   \zf@update@family{bfit feat:#1}
1242 }
1243 \define@key[zf]{preparse}{SlantedFeatures}{
1244   \def\zf@sl@feat{, #1}
1245   \zf@update@family{sl feat:#1}
1246 }
1247 \define@key[zf]{preparse}{BoldSlantedFeatures}{
1248   \def\zf@bfs sl@feat{, #1}
1249   \zf@update@family{bfs sl feat:#1}
1250 }
```

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

```
1251 \define@key[zf]{options}{SmallCapsFeatures}{
1252   \unless\ifzf@firsttime\def\zf@sc@feat{, #1}\fi
1253   \zf@update@family{sc feat:\zap@space #1~\@empty}
1254 }
```

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

```
1255 \define@key[zf]{preparse}{SizeFeatures}{
1256   \tl_set:Nn \l_fontspec_sizefeat_clist {#1}
1257   \zf@update@family{sizefeat:\zap@space #1~\@empty}
1258 }
1259 \define@key[zf]{sizing}{Size}{ \tl_set:Nn \l_fontspec_size_tl {#1} }
1260 \define@key[zf]{sizing}{Font}{
1261   \fontspec_complete_fontname:Nn \l_fontspec_sizedfont_tl {#1}
1262 }
```

23.10.4 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.

```

1263 \define@key[zf]{options}{Scale}{
1264   \prg_case_str:nnn {#1}
1265   {
1266     {MatchLowercase} { \fontspec_calc_scale:n {5} }
1267     {MatchUppercase} { \fontspec_calc_scale:n {8} }
1268   }
1269   { \tl_set:Nx \l_fontspec_scale_tl {#1} }
1270   \zf@update@family{+scale:\l_fontspec_scale_tl}
1271   \tl_set:Nx \l_fontspec_scale_tl { s*[\l_fontspec_scale_tl] }
1272 }

```

`\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_{TeX}).

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

```

1273 \cs_new:Npn \fontspec_calc_scale:n #1 {
1274   \group_begin:
1275   \rmfamily
1276   \fontspec_set_font_dimen:NnN \@tempdima {#1} \font
1277   \fontspec_set_font_dimen:NnN \@tempdimb {#1} \zf@basefont
1278   \dim_set:Nn \@tempdimc { 1pt*\@tempdima/\@tempdimb }
1279   \tl_gset:Nx \l_fontspec_scale_tl {\strip@pt\@tempdimc}
1280   \fontspec_info:n {set-scale}
1281   \group_end:
1282 }

```

`\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 .tfm 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’.

```

1283 \cs_new:Npn \fontspec_set_font_dimen:NnN #1#2#3
1284 {
1285   \dim_set:Nn #1 { \fontdimen #2 #3 }
1286   \dim_compare:nNnT #1 = {0pt} {
1287     \settoheight #1 {
1288       \tl_if_eq:nnTF {#3} {\font} \rmfamily #3
1289       \prg_case_int:nnn #2 {
1290         {5} {x} % x-height
1291         {8} {X} % cap-height
1292       } {?} % "else" clause; never reached.
1293     }
1294   }
1295 }

```

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

```

1296 \define@key[zf]{options}{WordSpace}{
1297   \zf@update@family{+wordspace:#1}
1298   \unless\ifzf@firsttime
1299     \zf@wordspace@parse#1,\zf@ii,\zf@iii,\zf@
1300   \fi
1301 }

```

`\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}`.

```

1302 \def\zf@wordspace@parse#1,#2,#3,#4\zf@{
1303   \def\@tempa{#4}
1304   \ifx\@tempa\@empty
1305     \setlength\@tempdima{#1\fontdimen2\zf@basefont}
1306     \@tempdimb\@tempdima
1307     \@tempdimc\@tempdima
1308   \else
1309     \setlength\@tempdima{#1\fontdimen2\zf@basefont}
1310     \setlength\@tempdimb{#2\fontdimen3\zf@basefont}
1311     \setlength\@tempdimc{#3\fontdimen4\zf@basefont}
1312   \fi
1313   \edef\zf@adjust{
1314     \zf@adjust
1315     \fontdimen2\font\the\@tempdima
1316     \fontdimen3\font\the\@tempdimb
1317     \fontdimen4\font\the\@tempdimc
1318   }
1319 }

```

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

```

1320 \define@key[zf]{options}{PunctuationSpace}{
1321   \zf@update@family{+punctspace:#1}
1322   \setlength\@tempdima{#1\fontdimen7\zf@basefont}
1323   \edef\zf@adjust{\zf@adjust\fontdimen7\font\the\@tempdima}
1324 }

```

Letterspacing

```

1325 \define@key[zf]{options}{LetterSpace}{
1326   \zf@update@family{+tracking:#1}
1327   \zf@update@ff{letterspace=#1}
1328 }

```

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.

```

1329 \define@key[zf]{options}{HyphenChar}{
1330   \zf@update@family{+hyphenchar:#1}
1331   \tl_if_eq:nnTF {#1} {None}
1332   {
1333     \tl_put_right:Nn \zf@adjust { \hyphenchar \font = \c_minus_one }

```

```

1334 }
1335 {
1336   \tl_if_single:nTF {#1}
1337     { \tl_set:Nn \l_fontspec_hyphenchar_tl {'#1} }
1338     { \tl_set:Nn \l_fontspec_hyphenchar_tl { #1} }
1339   \fontspec_glyph_if_exist:NnTF \zf@basefont {\l_fontspec_hyphenchar_tl}
1340   {
1341     \xetex_or luatex:nnn { \tl_put_right:Nn \zf@adjust }
1342     { \hyphenchar \font = \l_fontspec_hyphenchar_tl \scan_stop: }
1343     {
1344       \hyphenchar \font = \c_zero
1345       \luatexprehyphenchar = \l_fontspec_hyphenchar_tl \scan_stop:
1346     }
1347   }
1348   { \fontspec_error:nx {no-glyph}{#1} }
1349 }
1350 }

```

Color

```

1351 \define@key[zf]{options}{Color}{
1352   \zf@update@family{+col:#1}
1353   \cs_if_exist:cTF {\token_to_str:N\color@#1}
1354   {
1355     \convertcolorspec{named}{#1}{HTML}\l_fontspec_hexcol_tl
1356   }
1357   {
1358     \intexpr_compare:nTF { \tl_elt_count:n {#1} == 6 }
1359     { \tl_set:Nn \l_fontspec_hexcol_tl {#1} }
1360     {
1361       \intexpr_compare:nTF { \tl_elt_count:n {#1} == 8 }
1362       { \fontspec_parse_colour:viii #1 }
1363       {
1364         \ifzf@firsttime\else
1365           \fontspec_warning:nx {bad-colour} {#1}
1366         \fi
1367       }
1368     }
1369   }
1370 }
1371 \cs_set:Npn \fontspec_parse_colour:viii #1#2#3#4#5#6#7#8 {
1372   \tl_set:Nn \l_fontspec_hexcol_tl {#1#2#3#4#5#6}
1373   \tl_if_eq:NNF \l_fontspec_opacity_tl \g_fontspec_opacity_tl
1374   {
1375     \ifzf@firsttime\else
1376       \fontspec_warning:nx {opa-twice-col} {#7#8}
1377     \fi
1378   }
1379   \tl_set:Nn \l_fontspec_opacity_tl {#7#8}
1380 }
1381 \keyval@alias@key[zf]{options}{Color}{Colour}
1382 \newcounter{fontspec_tmp_int}

```

```

1383 \define@key[zf]{options}{Opacity}{
1384   \zf@update@family{+opac:#1}
1385   \setcounter {fontspec_tmp_int} { 255*\real{#1} }
1386   \tl_if_eq:NNF \l_fontspec_opacity_tl \g_fontspec_opacity_tl
1387   {
1388     \ifzf@firsttime\else
1389       \fontspec_warning:nx {opa-twice} {#1}
1390     \fi
1391   }
1392   \tl_set:Nx \l_fontspec_opacity_tl
1393   { \hex2 { \value{fontspec_tmp_int} } }
1394 }

```

Mapping

```

1395 \xetex_or luatex:nnn {
1396   \define@key[zf]{options}{Mapping}
1397 }{
1398   \zf@update@family{+map:#1}
1399   \zf@update@ff{mapping=#1}
1400 }{
1401   \tl_if_eq:nnTF {#1} {tex-text} {
1402     \fontspec_warning:n {no-mapping-ligtex}
1403     \msg_redirect_name:nnn {fontspec} {no-mapping-ligtex} {none}
1404     \setkeys[zf]{options}{ Ligatures=TeX }
1405   }{
1406     \fontspec_warning:n {no-mapping}
1407   }
1408 }

```

FeatureFile

```

1409 \define@key[zf]{options}{FeatureFile}{
1410   \zf@update@family{+fea:#1}
1411   \zf@update@ff{featurefile=#1}
1412 }

```

23.10.5 Continuous font axes

```

1413 \define@key[zf]{options}{Weight}{
1414   \zf@update@family{+weight:#1}
1415   \zf@update@ff{weight=#1}
1416 }
1417 \define@key[zf]{options}{Width}{
1418   \zf@update@family{+width:#1}
1419   \zf@update@ff{width=#1}
1420 }
1421 \define@key[zf]{options}{OpticalSize}{
1422   \xetex_or luatex:nn {
1423     \ifzf@icu
1424       \tl_set:Nn \l_fontspec_optical_size_tl {/ S = #1}
1425       \zf@update@family{+size:#1}

```

```

1426 \fi
1427 \ifzf@mm
1428   \zf@update@family{+size:#1}
1429   \zf@update@ff{optical size=#1}
1430 \fi
1431 \ifzf@icu\else
1432   \ifzf@mm\else
1433     \ifzf@firsttime
1434       \fontspec_warning:n {no-opticals}
1435     \fi
1436   \fi
1437 \fi
1438 }{
1439   \tl_set:Nn \l_fontspec_optical_size_tl {/ S = #1}
1440   \zf@update@family{+size:#1}
1441 }
1442 }

```

23.10.6 Font transformations

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

```

1443 \define@key[zf]{options}{FakeSlant}[0.2]{
1444   \zf@update@family{+slant:#1}
1445   \zf@update@ff{slant=#1}
1446 }
1447 \define@key[zf]{options}{FakeStretch}[1.2]{
1448   \zf@update@family{+extend:#1}
1449   \zf@update@ff{extend=#1}
1450 }
1451 \define@key[zf]{options}{FakeBold}[1.5]{
1452   \zf@update@family{+embolden:#1}
1453   \zf@update@ff{embolden=#1}
1454 }

```

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)

```

1455 \define@key[zf]{options}{AutoFakeSlant}[0.2]{
1456   \ifzf@firsttime
1457     \tl_set:Nn \zf@fake@slant {#1}
1458     \tl_put_right:Nn \zf@it@feat {,FakeSlant=#1}
1459     \tl_set_eq:NN \zf@it \zf@fontname
1460     \zf@update@family{fakeit:#1}
1461     \tl_if_empty:NF \zf@fake@embolden {
1462       \tl_put_right:Nx \zf@bfit@feat
1463       {,FakeBold=\zf@fake@embolden,FakeSlant=#1}
1464       \tl_set_eq:NN \zf@bfit \zf@fontname
1465     }

```



```

1466 \fi
1467 }

Same but reversed:
1468 \define@key[zf]{options}{AutoFakeBold}[1.5]{
1469   \ifzf@firsttime
1470     \tl_set:Nn \zf@fake@embolden {#1}
1471     \tl_put_right:Nn \zf@bf@feat {,FakeBold=#1}
1472     \tl_set_eq:NN \zf@bf \zf@fontname
1473     \zf@update@family{fakebf:#1}
1474     \tl_if_empty:NF \zf@fake@slant {
1475       \tl_put_right:Nx \zf@bf@feat
1476         {,FakeSlant=\zf@fake@slant,FakeBold=#1}
1477       \tl_set_eq:NN \zf@bf@it \zf@fontname
1478     }
1479 \fi
1480 }

```

23.10.7 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.

```

1481 \zf@define@font@feature{Ligatures}
1482 \zf@define@feature@option{Ligatures}{Required}      {1}{0}{+rlig}
1483 \zf@define@feature@option{Ligatures}{NoRequired}    {1}{1}{-rlig}
1484 \zf@define@feature@option{Ligatures}{Common}        {1}{2}{+liga}
1485 \zf@define@feature@option{Ligatures}{NoCommon}      {1}{3}{-liga}
1486 \zf@define@feature@option{Ligatures}{Rare}          {1}{4}{+dlig}
1487 \zf@define@feature@option{Ligatures}{NoRare}        {1}{5}{-dlig}
1488 \zf@define@feature@option{Ligatures}{Discretionary} {1}{4}{+dlig}
1489 \zf@define@feature@option{Ligatures}{NoDiscretionary}{1}{5}{-dlig}
1490 \zf@define@feature@option{Ligatures}{Contextual}    {}{} {+clig}
1491 \zf@define@feature@option{Ligatures}{NoContextual}  {}{} {-clig}
1492 \zf@define@feature@option{Ligatures}{Historic}      {}{} {+hlig}
1493 \zf@define@feature@option{Ligatures}{NoHistoric}    {}{} {-hlig}
1494 \zf@define@feature@option{Ligatures}{Logos}         {1}{6} {}
1495 \zf@define@feature@option{Ligatures}{NoLogos}       {1}{7} {}
1496 \zf@define@feature@option{Ligatures}{Rebus}         {1}{8} {}
1497 \zf@define@feature@option{Ligatures}{NoRebus}       {1}{9} {}
1498 \zf@define@feature@option{Ligatures}{Diphthong}     {1}{10} {}
1499 \zf@define@feature@option{Ligatures}{NoDiphthong}   {1}{11} {}
1500 \zf@define@feature@option{Ligatures}{Squared}       {1}{12} {}
1501 \zf@define@feature@option{Ligatures}{NoSquared}     {1}{13} {}
1502 \zf@define@feature@option{Ligatures}{AbbrevSquared} {1}{14} {}
1503 \zf@define@feature@option{Ligatures}{NoAbbrevSquared}{1}{15} {}
1504 \zf@define@feature@option{Ligatures}{Icelandic}     {1}{32} {}
1505 \zf@define@feature@option{Ligatures}{NoIcelandic}   {1}{33} {}

```

Emulate CM extra ligatures.

```

1506 \define@key[zf@feat]{Ligatures}{TeX}[]{}
1507 \xetex_or luatex:nn {

```

```

1508 \zf@update@family{+map:tex-text}
1509 \zf@update@ff{mapping=tex-text}
1510 }{
1511 \zf@update@family{+tlig+trep}
1512 \zf@update@ff{+tlig;trep}
1513 }
1514 }

```

23.10.8 Letters

```

1515 \zf@define@font@feature{Letters}
1516 \zf@define@feature@option{Letters}{Normal} {3}{0}{}
1517 \zf@define@feature@option{Letters}{Uppercase} {3}{1}{+case}
1518 \zf@define@feature@option{Letters}{Lowercase} {3}{2}{}
1519 \zf@define@feature@option{Letters}{SmallCaps} {3}{3}{+smcp}
1520 \zf@define@feature@option{Letters}{PetiteCaps} {} {} {+pcap}
1521 \zf@define@feature@option{Letters}{UppercaseSmallCaps} {} {} {+c2sc}
1522 \zf@define@feature@option{Letters}{UppercasePetiteCaps} {} {} {+c2pc}
1523 \zf@define@feature@option{Letters}{InitialCaps} {3}{4}{}
1524 \zf@define@feature@option{Letters}{Unicase} {} {} {+unic}

```

23.10.9 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.

```

1525 \zf@define@font@feature{Numbers}
1526 \zf@define@feature@option{Numbers}{Monospaced} {6} {0}{+tnum}
1527 \zf@define@feature@option{Numbers}{Proportional} {6} {1}{+pnum}
1528 \zf@define@feature@option{Numbers}{Lowercase} {21}{0}{+onum}
1529 \zf@define@feature@option{Numbers}{OldStyle} {21}{0}{+onum}
1530 \zf@define@feature@option{Numbers}{Uppercase} {21}{1}{+lnum}
1531 \zf@define@feature@option{Numbers}{Lining} {21}{1}{+lnum}
1532 \zf@define@feature@option{Numbers}{SlashedZero} {14}{5}{+zero}
1533 \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.

```

1534 \luaotload_if_engine:T {
1535 \zf@define@feature@option{Numbers}{Arabic}{}{}{+anum}
1536 }

```

23.10.10 Contextuals

```

1537 \zf@define@font@feature {Contextuals}
1538 \zf@define@feature@option{Contextuals}{Swash} {} {} {+cswsh}
1539 \zf@define@feature@option{Contextuals}{NoSwash} {} {} {-cswsh}
1540 \zf@define@feature@option{Contextuals}{Alternate} {} {} {+calt}
1541 \zf@define@feature@option{Contextuals}{NoAlternate} {} {} {-calt}
1542 \zf@define@feature@option{Contextuals}{WordInitial} {8}{0}{+init}
1543 \zf@define@feature@option{Contextuals}{NoWordInitial} {8}{1}{-init}

```

```

1544 \zf@define@feature@option{Contextuals}{WordFinal}      {8}{2}{+fina}
1545 \zf@define@feature@option{Contextuals}{NoWordFinal}     {8}{3}{-fina}
1546 \zf@define@feature@option{Contextuals}{LineInitial}     {8}{4}{ }
1547 \zf@define@feature@option{Contextuals}{NoLineInitial}   {8}{5}{ }
1548 \zf@define@feature@option{Contextuals}{LineFinal}       {8}{6}{+falt}
1549 \zf@define@feature@option{Contextuals}{NoLineFinal}     {8}{7}{-falt}
1550 \zf@define@feature@option{Contextuals}{Inner}            {8}{8}{+medi}
1551 \zf@define@feature@option{Contextuals}{NoInner}          {8}{9}{-medi}

```

23.10.11 Diacritics

```

1552 \zf@define@font@feature{Diacritics}
1553 \zf@define@feature@option{Diacritics}{Show}              {9}{0}{ }
1554 \zf@define@feature@option{Diacritics}{Hide}              {9}{1}{ }
1555 \zf@define@feature@option{Diacritics}{Decompose}         {9}{2}{ }
1556 \zf@define@feature@option{Diacritics}{MarkToBase}        { }{ }{+mark}
1557 \zf@define@feature@option{Diacritics}{NoMarkToBase}       { }{ }{-mark}
1558 \zf@define@feature@option{Diacritics}{MarkToMark}         { }{ }{+mkmk}
1559 \zf@define@feature@option{Diacritics}{NoMarkToMark}       { }{ }{-mkmk}
1560 \zf@define@feature@option{Diacritics}{AboveBase}          { }{ }{+abvm}
1561 \zf@define@feature@option{Diacritics}{NoAboveBase}        { }{ }{-abvm}
1562 \zf@define@feature@option{Diacritics}{BelowBase}          { }{ }{+blwm}
1563 \zf@define@feature@option{Diacritics}{NoBelowBase}        { }{ }{-blwm}

```

23.10.12 Kerning

```

1564 \zf@define@font@feature{Kerning}
1565 \zf@define@feature@option{Kerning}{Uppercase}             { }{ }{+csp}
1566 \zf@define@feature@option{Kerning}{On}                    { }{ }{+kern}
1567 \zf@define@feature@option{Kerning}{Off}                   { }{ }{-kern}
1568 %\zf@define@feature@option{Kerning}{Vertical}             { }{ }{+vkern}
1569 %\zf@define@feature@option{Kerning}
1570 % {VerticalAlternateProportional} { }{ }{+vpal}
1571 %\zf@define@feature@option{Kerning}{VerticalAlternateHalfWidth} { }{ }{+vhal}

```

23.10.13 Vertical position

```

1572 \zf@define@font@feature{VerticalPosition}
1573 \zf@define@feature@option{VerticalPosition}{Normal}       {10}{0}{ }
1574 \zf@define@feature@option{VerticalPosition}{Superior}     {10}{1}{+sup}
1575 \zf@define@feature@option{VerticalPosition}{Inferior}     {10}{2}{+sub}
1576 \zf@define@feature@option{VerticalPosition}{Ordinal}      {10}{3}{+ordn}
1577 \zf@define@feature@option{VerticalPosition}{Numerator}    { } { }{+numr}
1578 \zf@define@feature@option{VerticalPosition}{Denominator}  { } { }{+dnom}
1579 \zf@define@feature@option{VerticalPosition}{ScientificInferior} { }{ }{+sinf}

```

23.10.14 Fractions

```

1580 \zf@define@font@feature{Fractions}
1581 \zf@define@feature@option{Fractions}{On}                   {11}{1}{+frac}
1582 \zf@define@feature@option{Fractions}{Off}                  {11}{0}{-frac}
1583 \zf@define@feature@option{Fractions}{Diagonal}             {11}{2}{ }
1584 \zf@define@feature@option{Fractions}{Alternate}            { } { }{+afrc}

```

23.10.15 Alternates and variants

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

```
1585 \define@key[zf]{options}{Alternate}[0]{
1586   \clist_set_eq:NN \l_fontspec_tmpa_clist \XKV@rm
1587   \setkeys*[zf@feat]{Alternate}{#1}
1588   \unless\ifx\XKV@rm\@empty
1589     \def\XKV@tfam{Alternate}
1590     \fontspec_make_feature:nnx {17}{#1} { \fontspec_salt:n {#1} }
1591   \fi
1592   \clist_set_eq:NN \XKV@rm \l_fontspec_tmpa_clist
1593 }

1594 \xetex_or luatex:nnn { \cs_set:Npn \fontspec_salt:n #1 }
1595 { +salt=#1 } { +salt= \intexpr_eval:n {#1+1} }

1596 \define@key[zf]{options}{Variant}{
1597   \clist_set_eq:NN \l_fontspec_tmpa_clist \XKV@rm
1598   \setkeys*[zf@feat]{Variant}{#1}
1599   \unless\ifx\XKV@rm\@empty
1600     \def\XKV@tfam{Variant}
1601     \fontspec_make_feature:nnx {18}{#1} { +ss \two@digits {#1} }
1602   \fi
1603   \clist_set_eq:NN \XKV@rm \l_fontspec_tmpa_clist
1604 }

1605 \aliasfontfeature{Variant}{StylisticSet}

1606 \define@key[zf]{options}{CharacterVariant}{
1607   \clist_set_eq:NN \l_fontspec_tmpa_clist \XKV@rm
1608   \setkeys*[zf@feat]{CharacterVariant}{#1}
1609   \unless\ifx\XKV@rm\@empty
1610     \def\XKV@tfam{CharacterVariant}
1611     \fontspec_make_feature:nnx {}{} { +cv \two@digits {#1} }
1612   \fi
1613   \clist_set_eq:NN \XKV@rm \l_fontspec_tmpa_clist
1614 }
```

23.10.16 Style

```
1615 \zf@define@font@feature{Style}
1616 \zf@define@feature@option{Style}{Alternate}      {} {} {+salt}
1617 \zf@define@feature@option{Style}{Italic}          {32}{2}{+ital}
1618 \zf@define@feature@option{Style}{Ruby}           {28}{2}{+ruby}
1619 \zf@define@feature@option{Style}{Swash}          {} {} {+swsh}
1620 \zf@define@feature@option{Style}{Historic}        {} {} {+hist}
1621 \zf@define@feature@option{Style}{Display}         {19}{1}{ }
1622 \zf@define@feature@option{Style}{Engraved}        {19}{2}{ }
1623 \zf@define@feature@option{Style}{TitlingCaps}     {19}{4}{+titl}
1624 \zf@define@feature@option{Style}{TallCaps}        {19}{5}{ }
1625 \zf@define@feature@option{Style}{HorizontalKana} {} {} {+hkna}
1626 \zf@define@feature@option{Style}{VerticalKana}   {} {} {+vkna}
```

23.10.17 CJK shape

```

1627 \zf@define@font@feature{CJKShape}
1628 \zf@define@feature@option{CJKShape}{Traditional}{20}{0}{+trad}
1629 \zf@define@feature@option{CJKShape}{Simplified}{20}{1}{+smp1}
1630 \zf@define@feature@option{CJKShape}{JIS1978}{20}{2}{+jp78}
1631 \zf@define@feature@option{CJKShape}{JIS1983}{20}{3}{+jp83}
1632 \zf@define@feature@option{CJKShape}{JIS1990}{20}{4}{+jp90}
1633 \zf@define@feature@option{CJKShape}{Expert}{20}{10}{+expt}
1634 \zf@define@feature@option{CJKShape}{NLC}{20}{13}{+nlck}

```

23.10.18 Character width

```

1635 \zf@define@font@feature{CharacterWidth}
1636 \zf@define@feature@option{CharacterWidth}{Proportional}{22}{0}{+pwid}
1637 \zf@define@feature@option{CharacterWidth}{Full}{22}{1}{+fwid}
1638 \zf@define@feature@option{CharacterWidth}{Half}{22}{2}{+hwid}
1639 \zf@define@feature@option{CharacterWidth}{Third}{22}{3}{+twid}
1640 \zf@define@feature@option{CharacterWidth}{Quarter}{22}{4}{+qwid}
1641 \zf@define@feature@option{CharacterWidth}{AlternateProportional}{22}{5}{+palt}
1642 \zf@define@feature@option{CharacterWidth}{AlternateHalf}{22}{6}{+halt}
1643 \zf@define@feature@option{CharacterWidth}{Default}{22}{7}{ }

```

23.10.19 Annotation

```

1644 \zf@define@feature@option{Annotation}{Off}{24}{0}{ }
1645 \zf@define@feature@option{Annotation}{Box}{24}{1}{ }
1646 \zf@define@feature@option{Annotation}{RoundedBox}{24}{2}{ }
1647 \zf@define@feature@option{Annotation}{Circle}{24}{3}{ }
1648 \zf@define@feature@option{Annotation}{BlackCircle}{24}{4}{ }
1649 \zf@define@feature@option{Annotation}{Parenthesis}{24}{5}{ }
1650 \zf@define@feature@option{Annotation}{Period}{24}{6}{ }
1651 \zf@define@feature@option{Annotation}{RomanNumerals}{24}{7}{ }
1652 \zf@define@feature@option{Annotation}{Diamond}{24}{8}{ }
1653 \zf@define@feature@option{Annotation}{BlackSquare}{24}{9}{ }
1654 \zf@define@feature@option{Annotation}{BlackRoundSquare}{24}{10}{ }
1655 \zf@define@feature@option{Annotation}{DoubleCircle}{24}{11}{ }

1656 \define@key{zf}[options]{Annotation}[0]{
1657   \clist_set_eq:NN \l_fontspec_tmpa_clist \XKV@rm
1658   \setkeys*{zf@feat}{Annotation}{#1}
1659   \unless\ifx\XKV@rm\@empty
1660     \def\XKV@tfam{Alternate}
1661     \fontspec_make_feature:nxx { } { { \fontspec_nalt:n {#1} } }
1662   \fi
1663   \clist_set_eq:NN \XKV@rm \l_fontspec_tmpa_clist
1664 }

1665 \xetex_or luatex:nnn { \cs_set:Npn \fontspec_nalt:n #1 }
1666 {+nalt=#1} { +nalt= \intexpr_eval:n {#1+1} }

```

23.10.20 Vertical

```

1667 \zf@define@font@feature{Vertical}
1668 \define@key{zf@feat}{Vertical}{RotatedGlyphs}[] {
1669   \ifzf@icu
1670     \fontspec_make_feature:nnn { } {+vrt2}
1671     \zf@update@family{+vert}

```

```

1672 \zf@update@ff{vertical}
1673 \else
1674 \zf@update@family{+vert}
1675 \zf@update@ff{vertical}
1676 \fi
1677 }

```

23.10.21 Script

```

1678 \newfontscript{Arabic}{arab} \newfontscript{Armenian}{armn}
1679 \newfontscript{Balinese}{bali} \newfontscript{Bengali}{beng}
1680 \newfontscript{Bopomofo}{bopo} \newfontscript{Braille}{brai}
1681 \newfontscript{Buginese}{bugi} \newfontscript{Buhid}{buhd}
1682 \newfontscript{Byzantine~Music}{byzm} \newfontscript{Canadian~Syllabics}{cans}
1683 \newfontscript{Cherokee}{cher}
1684 \newfontscript{CJK~Ideographic}{hani} \newfontscript{Coptic}{copt}
1685 \newfontscript{Cypriot~Syllabary}{cpri} \newfontscript{Cyrillic}{cyr1}
1686 \newfontscript{Default}{DFLT} \newfontscript{Deseret}{dsrt}
1687 \newfontscript{Devanagari}{deva} \newfontscript{Ethiopic}{ethi}
1688 \newfontscript{Georgian}{geor} \newfontscript{Glagolitic}{glag}
1689 \newfontscript{Gothic}{goth} \newfontscript{Greek}{grek}
1690 \newfontscript{Gujarati}{gujr} \newfontscript{Gurmukhi}{guru}
1691 \newfontscript{Hangul~Jamo}{jamo} \newfontscript{Hangul}{hang}
1692 \newfontscript{Hanunoo}{hano} \newfontscript{Hebrew}{hebr}
1693 \newfontscript{Hiragana~and~Katakana}{kana}
1694 \newfontscript{Javanese}{java} \newfontscript{Kannada}{knda}
1695 \newfontscript{Kharosthi}{khar} \newfontscript{Khmer}{khmr}
1696 \newfontscript{Lao}{lao~} \newfontscript{Latin}{latn}
1697 \newfontscript{Limbu}{limb} \newfontscript{Linear~B}{linb}
1698 \newfontscript{Malayalam}{mlym} \newfontscript{Math}{math}
1699 \newfontscript{Mongolian}{mong}
1700 \newfontscript{Musical~Symbols}{musc} \newfontscript{Myanmar}{mymr}
1701 \newfontscript{N'ko}{nko~} \newfontscript{Ogham}{ogam}
1702 \newfontscript{Old~Italic}{ital}
1703 \newfontscript{Old~Persian~Cuneiform}{xpeo}
1704 \newfontscript{Oriya}{orya} \newfontscript{Osmanya}{osma}
1705 \newfontscript{Phags-pa}{phag} \newfontscript{Phoenician}{phnx}
1706 \newfontscript{Runic}{runr} \newfontscript{Shavian}{shaw}
1707 \newfontscript{Sinhala}{sinh}
1708 \newfontscript{Sumero-Akkadian~Cuneiform}{xsux}
1709 \newfontscript{Syloti~Nagri}{sylo} \newfontscript{Syriac}{syr1}
1710 \newfontscript{Tagalog}{tglg} \newfontscript{Tagbanwa}{tagb}
1711 \newfontscript{Tai~Le}{tale} \newfontscript{Tai~Lu}{tal1}
1712 \newfontscript{Tamil}{taml} \newfontscript{Telugu}{telu}
1713 \newfontscript{Thaana}{thaa} \newfontscript{Thai}{thai}
1714 \newfontscript{Tibetan}{tibt} \newfontscript{Tifinagh}{tfng}
1715 \newfontscript{Ugaritic~Cuneiform}{ugar} \newfontscript{Yi}{yi~}

```

For convenience:

```

1716 \newfontscript{Kana}{kana}
1717 \newfontscript{Maths}{math}
1718 \newfontscript{CJK}{hani}

```

23.10.22 Language

1719 \newfontlanguage{Abaza}{ABA}\newfontlanguage{Abkhazian}{ABK}
1720 \newfontlanguage{Adyghe}{ADY}\newfontlanguage{Afrikaans}{AFK}
1721 \newfontlanguage{Afar}{AFR}\newfontlanguage{Agaw}{AGW}
1722 \newfontlanguage{Altai}{ALT}\newfontlanguage{Amharic}{AMH}
1723 \newfontlanguage{Arabic}{ARA}\newfontlanguage{Aari}{ARI}
1724 \newfontlanguage{Arakanese}{ARK}\newfontlanguage{Assamese}{ASM}
1725 \newfontlanguage{Athapaskan}{ATH}\newfontlanguage{Avar}{AVR}
1726 \newfontlanguage{Awadhi}{AWA}\newfontlanguage{Aymara}{AYM}
1727 \newfontlanguage{Azeri}{AZE}\newfontlanguage{Badaga}{BAD}
1728 \newfontlanguage{Baghelkhandi}{BAG}\newfontlanguage{Balkar}{BAL}
1729 \newfontlanguage{Baule}{BAU}\newfontlanguage{Berber}{BBR}
1730 \newfontlanguage{Bench}{BCH}\newfontlanguage{Bible~Cree}{BCR}
1731 \newfontlanguage{Belarussian}{BEL}\newfontlanguage{Bemba}{BEM}
1732 \newfontlanguage{Bengali}{BEN}\newfontlanguage{Bulgarian}{BGR}
1733 \newfontlanguage{Bhili}{BHI}\newfontlanguage{Bhojpuri}{BHO}
1734 \newfontlanguage{Bikol}{BIK}\newfontlanguage{Bilen}{BIL}
1735 \newfontlanguage{Blackfoot}{BKF}\newfontlanguage{Balochi}{BLI}
1736 \newfontlanguage{Balante}{BLN}\newfontlanguage{Balti}{BLT}
1737 \newfontlanguage{Bambara}{BMB}\newfontlanguage{Bamileke}{BML}
1738 \newfontlanguage{Breton}{BRE}\newfontlanguage{Brahui}{BRH}
1739 \newfontlanguage{Braj~Bhasha}{BRI}\newfontlanguage{Burmese}{BRM}
1740 \newfontlanguage{Bashkir}{BSH}\newfontlanguage{Beti}{BTI}
1741 \newfontlanguage{Catalan}{CAT}\newfontlanguage{Cebuano}{CEB}
1742 \newfontlanguage{Chechen}{CHE}\newfontlanguage{Chaha~Gurage}{CHG}
1743 \newfontlanguage{Chattisgarhi}{CHH}\newfontlanguage{Chichewa}{CHI}
1744 \newfontlanguage{Chukchi}{CHK}\newfontlanguage{Chipewyan}{CHP}
1745 \newfontlanguage{Cherokee}{CHR}\newfontlanguage{Chuvash}{CHU}
1746 \newfontlanguage{Comorian}{CMR}\newfontlanguage{Coptic}{COP}
1747 \newfontlanguage{Cree}{CRE}\newfontlanguage{Carrier}{CRR}
1748 \newfontlanguage{Crimean~Tatar}{CRT}\newfontlanguage{Church~Slavonic}{CSL}
1749 \newfontlanguage{Czech}{CSY}\newfontlanguage{Danish}{DAN}
1750 \newfontlanguage{Dargwa}{DAR}\newfontlanguage{Woods~Cree}{DCR}
1751 \newfontlanguage{German}{DEU}
1752 \newfontlanguage{Dogri}{DGR}\newfontlanguage{Divehi}{DIV}
1753 \newfontlanguage{Djerma}{DJR}\newfontlanguage{Dangme}{DNG}
1754 \newfontlanguage{Dinka}{DNK}\newfontlanguage{Dungan}{DUN}
1755 \newfontlanguage{Dzongkha}{DZN}\newfontlanguage{Ebira}{EBI}
1756 \newfontlanguage{Eastern~Cree}{ECR}\newfontlanguage{Edo}{EDO}
1757 \newfontlanguage{Efik}{EFI}\newfontlanguage{Greek}{ELL}
1758 \newfontlanguage{English}{ENG}\newfontlanguage{Erzya}{ERZ}
1759 \newfontlanguage{Spanish}{ESP}\newfontlanguage{Estonian}{ETI}
1760 \newfontlanguage{Basque}{EUQ}\newfontlanguage{Evenki}{EVK}
1761 \newfontlanguage{Even}{EVN}\newfontlanguage{Ewe}{EWE}
1762 \newfontlanguage{French~Antillean}{FAN}
1763 \newfontlanguage{Farsi}{FAR}
1764 \newfontlanguage{Parsi}{FAR}
1765 \newfontlanguage{Persian}{FAR}
1766 \newfontlanguage{Finnish}{FIN}\newfontlanguage{Fijian}{FJI}
1767 \newfontlanguage{Flemish}{FLE}\newfontlanguage{Forest~Nenets}{FNE}
1768 \newfontlanguage{Fon}{FON}\newfontlanguage{Faroese}{FOS}

1769 \newfontlanguage{French}{FRA}\newfontlanguage{Frisian}{FRI}
1770 \newfontlanguage{Friulian}{FRL}\newfontlanguage{Futa}{FTA}
1771 \newfontlanguage{Fulani}{FUL}\newfontlanguage{Ga}{GAD}
1772 \newfontlanguage{Gaelic}{GAE}\newfontlanguage{Gagauz}{GAG}
1773 \newfontlanguage{Galician}{GAL}\newfontlanguage{Garshuni}{GAR}
1774 \newfontlanguage{Garhwali}{GAW}\newfontlanguage{Ge'ez}{GEZ}
1775 \newfontlanguage{Gilyak}{GIL}\newfontlanguage{Gumuz}{GMZ}
1776 \newfontlanguage{Gondi}{GON}\newfontlanguage{Greenlandic}{GRN}
1777 \newfontlanguage{Garo}{GRO}\newfontlanguage{Guarani}{GUA}
1778 \newfontlanguage{Gujarati}{GUJ}\newfontlanguage{Haitian}{HAI}
1779 \newfontlanguage{Halam}{HAL}\newfontlanguage{Harauti}{HAR}
1780 \newfontlanguage{Hausa}{HAU}\newfontlanguage{Hawaii}{HAW}
1781 \newfontlanguage{Hammer-Banna}{HBN}\newfontlanguage{Hiligaynon}{HIL}
1782 \newfontlanguage{Hindi}{HIN}\newfontlanguage{High~Mari}{HMA}
1783 \newfontlanguage{Hindko}{HND}\newfontlanguage{Ho}{HO}
1784 \newfontlanguage{Harari}{HRI}\newfontlanguage{Croatian}{HRV}
1785 \newfontlanguage{Hungarian}{HUN}\newfontlanguage{Armenian}{HYE}
1786 \newfontlanguage{Igbo}{IBO}\newfontlanguage{Ijo}{IJO}
1787 \newfontlanguage{Ilokano}{ILO}\newfontlanguage{Indonesian}{IND}
1788 \newfontlanguage{Ingush}{ING}\newfontlanguage{Inuktitut}{INU}
1789 \newfontlanguage{Irish}{IRI}\newfontlanguage{Irish~Traditional}{IRT}
1790 \newfontlanguage{Icelandic}{ISL}\newfontlanguage{Inari~Sami}{ISM}
1791 \newfontlanguage{Italian}{ITA}\newfontlanguage{Hebrew}{IWR}
1792 \newfontlanguage{Javanese}{JAV}\newfontlanguage{Yiddish}{JII}
1793 \newfontlanguage{Japanese}{JAN}\newfontlanguage{Judezmo}{JUD}
1794 \newfontlanguage{Jula}{JUL}\newfontlanguage{Kabardian}{KAB}
1795 \newfontlanguage{Kachchi}{KAC}\newfontlanguage{Kalenjin}{KAL}
1796 \newfontlanguage{Kannada}{KAN}\newfontlanguage{Karachay}{KAR}
1797 \newfontlanguage{Georgian}{KAT}\newfontlanguage{Kazakh}{KAZ}
1798 \newfontlanguage{Kebena}{KEB}\newfontlanguage{Khutsuri~Georgian}{KGE}
1799 \newfontlanguage{Khakass}{KHA}\newfontlanguage{Khanty-Kazim}{KHK}
1800 \newfontlanguage{Khmer}{KHM}\newfontlanguage{Khanty-Shurishkar}{KHS}
1801 \newfontlanguage{Khanty-Vakhi}{KHV}\newfontlanguage{Khowar}{KHW}
1802 \newfontlanguage{Kikuyu}{KIK}\newfontlanguage{Kirghiz}{KIR}
1803 \newfontlanguage{Kisii}{KIS}\newfontlanguage{Kokni}{KKN}
1804 \newfontlanguage{Kalmyk}{KLM}\newfontlanguage{Kamba}{KMB}
1805 \newfontlanguage{Kumaoni}{KMN}\newfontlanguage{Komo}{KMO}
1806 \newfontlanguage{Komso}{KMS}\newfontlanguage{Kanuri}{KNR}
1807 \newfontlanguage{Kodagu}{KOD}\newfontlanguage{Korean~Old~Hangul}{KOH}
1808 \newfontlanguage{Konkani}{KOK}\newfontlanguage{Kikongo}{KON}
1809 \newfontlanguage{Komi-Permyak}{KOP}\newfontlanguage{Korean}{KOR}
1810 \newfontlanguage{Komi-Zyrian}{KOZ}\newfontlanguage{Kpelle}{KPL}
1811 \newfontlanguage{Krio}{KRI}\newfontlanguage{Karakalpak}{KRR}
1812 \newfontlanguage{Karelian}{KRL}\newfontlanguage{Karaim}{KRM}
1813 \newfontlanguage{Karen}{KRN}\newfontlanguage{Korete}{KRT}
1814 \newfontlanguage{Kashmiri}{KSH}\newfontlanguage{Khasi}{KSI}
1815 \newfontlanguage{Kildin~Sami}{KSM}\newfontlanguage{Kui}{KUI}
1816 \newfontlanguage{Kulvi}{KUL}\newfontlanguage{Kumyk}{KUM}
1817 \newfontlanguage{Kurdish}{KUR}\newfontlanguage{Kurukh}{KUU}
1818 \newfontlanguage{Kuy}{KUY}\newfontlanguage{Koryak}{KYK}
1819 \newfontlanguage{Ladin}{LAD}\newfontlanguage{Lahuli}{LAH}

1820 \newfontlanguage{Lak}{LAK}\newfontlanguage{Lambani}{LAM}
 1821 \newfontlanguage{Lao}{LAO}\newfontlanguage{Latin}{LAT}
 1822 \newfontlanguage{Laz}{LAZ}\newfontlanguage{L-Cree}{LCR}
 1823 \newfontlanguage{Ladakhi}{LDK}\newfontlanguage{Lezgi}{LEZ}
 1824 \newfontlanguage{Lingala}{LIN}\newfontlanguage{Low~Mari}{LMA}
 1825 \newfontlanguage{Limbu}{LMB}\newfontlanguage{Lomwe}{LMW}
 1826 \newfontlanguage{Lower~Sorbian}{LSB}\newfontlanguage{Lule~Sami}{LSM}
 1827 \newfontlanguage{Lithuanian}{LTH}\newfontlanguage{Luba}{LUB}
 1828 \newfontlanguage{Luganda}{LUG}\newfontlanguage{Luhya}{LUH}
 1829 \newfontlanguage{Luo}{LUO}\newfontlanguage{Latvian}{LVI}
 1830 \newfontlanguage{Majang}{MAJ}\newfontlanguage{Makua}{MAK}
 1831 \newfontlanguage{Malayalam~Traditional}{MAL}\newfontlanguage{Mansi}{MAN}
 1832 \newfontlanguage{Marathi}{MAR}\newfontlanguage{Marwari}{MAW}
 1833 \newfontlanguage{Mbundu}{MBN}\newfontlanguage{Manchu}{MCH}
 1834 \newfontlanguage{Moose~Cree}{MCR}\newfontlanguage{Mende}{MDE}
 1835 \newfontlanguage{Me'en}{MEN}\newfontlanguage{Mizo}{MIZ}
 1836 \newfontlanguage{Macedonian}{MKD}\newfontlanguage{Male}{MLE}
 1837 \newfontlanguage{Malagasy}{MLG}\newfontlanguage{Malinke}{MLN}
 1838 \newfontlanguage{Malayalam~Reformed}{MLR}\newfontlanguage{Malay}{MLY}
 1839 \newfontlanguage{Mandinka}{MND}\newfontlanguage{Mongolian}{MNG}
 1840 \newfontlanguage{Manipuri}{MNI}\newfontlanguage{Maninka}{MNK}
 1841 \newfontlanguage{Manx~Gaelic}{MNX}\newfontlanguage{Moksha}{MOK}
 1842 \newfontlanguage{Moldavian}{MOL}\newfontlanguage{Mon}{MON}
 1843 \newfontlanguage{Moroccan}{MOR}\newfontlanguage{Maori}{MRI}
 1844 \newfontlanguage{Maithili}{MTH}\newfontlanguage{Maltese}{MTS}
 1845 \newfontlanguage{Mundari}{MUN}\newfontlanguage{Naga-Assamese}{NAG}
 1846 \newfontlanguage{Nanai}{NAN}\newfontlanguage{Naskapi}{NAS}
 1847 \newfontlanguage{N-Cree}{NCR}\newfontlanguage{Ndebele}{NDB}
 1848 \newfontlanguage{Ndonga}{NDG}\newfontlanguage{Nepali}{NEP}
 1849 \newfontlanguage{Newari}{NEW}\newfontlanguage{Nagari}{NGR}
 1850 \newfontlanguage{Norway~House~Cree}{NHC}\newfontlanguage{Nisi}{NIS}
 1851 \newfontlanguage{Niuean}{NIU}\newfontlanguage{Nkole}{NKL}
 1852 \newfontlanguage{N'ko}{NKO}\newfontlanguage{Dutch}{NLD}
 1853 \newfontlanguage{Nogai}{NOG}\newfontlanguage{Norwegian}{NOR}
 1854 \newfontlanguage{Northern~Sami}{NSM}\newfontlanguage{Northern~Tai}{NTA}
 1855 \newfontlanguage{Esperanto}{NTO}\newfontlanguage{Nynorsk}{NYN}
 1856 \newfontlanguage{Oji-Cree}{OCR}\newfontlanguage{Ojibway}{OBJ}
 1857 \newfontlanguage{Oriya}{ORI}\newfontlanguage{Oromo}{ORO}
 1858 \newfontlanguage{Ossetian}{OSS}\newfontlanguage{Palestinian~Aramaic}{PAA}
 1859 \newfontlanguage{Pali}{PAL}\newfontlanguage{Punjabi}{PAN}
 1860 \newfontlanguage{Palpa}{PAP}\newfontlanguage{Pashto}{PAS}
 1861 \newfontlanguage{Polytonic~Greek}{PGR}\newfontlanguage{Pilipino}{PIL}
 1862 \newfontlanguage{Palaung}{PLG}\newfontlanguage{Polish}{PLK}
 1863 \newfontlanguage{Provencal}{PRO}\newfontlanguage{Portuguese}{PTG}
 1864 \newfontlanguage{Chin}{QIN}\newfontlanguage{Rajasthani}{RAJ}
 1865 \newfontlanguage{R-Cree}{RCR}\newfontlanguage{Russian~Buriat}{RBU}
 1866 \newfontlanguage{Riang}{RIA}\newfontlanguage{Rhaeto-Romanic}{RMS}
 1867 \newfontlanguage{Romanian}{ROM}\newfontlanguage{Romany}{ROY}
 1868 \newfontlanguage{Rusyn}{RSY}\newfontlanguage{Ruanda}{RUA}
 1869 \newfontlanguage{Russian}{RUS}\newfontlanguage{Sadri}{SAD}
 1870 \newfontlanguage{Sanskrit}{SAN}\newfontlanguage{Santali}{SAT}

```

1871 \newfontlanguage{Sayisi}{SAY}\newfontlanguage{Sekota}{SEK}
1872 \newfontlanguage{Selkup}{SEL}\newfontlanguage{Sango}{SGO}
1873 \newfontlanguage{Shan}{SHN}\newfontlanguage{Sibe}{SIB}
1874 \newfontlanguage{Sidamo}{SID}\newfontlanguage{Silte~Gurage}{SIG}
1875 \newfontlanguage{Skolt~Sami}{SKS}\newfontlanguage{Slovak}{SKY}
1876 \newfontlanguage{Slavey}{SLA}\newfontlanguage{Slovenian}{SLV}
1877 \newfontlanguage{Somali}{SML}\newfontlanguage{Samoan}{SMO}
1878 \newfontlanguage{Sena}{SNA}\newfontlanguage{Sindhi}{SND}
1879 \newfontlanguage{Sinhalese}{SNH}\newfontlanguage{Soninke}{SNK}
1880 \newfontlanguage{Sodo~Gurage}{SOG}\newfontlanguage{Sotho}{SOT}
1881 \newfontlanguage{Albanian}{SQI}\newfontlanguage{Serbian}{SRB}
1882 \newfontlanguage{Saraiki}{SRK}\newfontlanguage{Serer}{SRR}
1883 \newfontlanguage{South~Slavey}{SSL}\newfontlanguage{Southern~Sami}{SSM}
1884 \newfontlanguage{Suri}{SUR}\newfontlanguage{Svan}{SVA}
1885 \newfontlanguage{Swedish}{SVE}\newfontlanguage{Swadaya~Aramaic}{SWA}
1886 \newfontlanguage{Swahili}{SWK}\newfontlanguage{Swazi}{SWZ}
1887 \newfontlanguage{Sutu}{SXT}\newfontlanguage{Syriac}{SYR}
1888 \newfontlanguage{Tabasaran}{TAB}\newfontlanguage{Tajiki}{TAJ}
1889 \newfontlanguage{Tamil}{TAM}\newfontlanguage{Tatar}{TAT}
1890 \newfontlanguage{TH-Cree}{TCR}\newfontlanguage{Telugu}{TEL}
1891 \newfontlanguage{Tongan}{TGN}\newfontlanguage{Tigre}{TGR}
1892 \newfontlanguage{Tigrinya}{TGY}\newfontlanguage{Thai}{THA}
1893 \newfontlanguage{Tahitian}{THT}\newfontlanguage{Tibetan}{TIB}
1894 \newfontlanguage{Turkmen}{TKM}\newfontlanguage{Temne}{TMN}
1895 \newfontlanguage{Tswana}{TNA}\newfontlanguage{Tundra~Nenets}{TNE}
1896 \newfontlanguage{Tonga}{TNG}\newfontlanguage{Todo}{TOD}
1897 \newfontlanguage{Tsonga}{TSG}\newfontlanguage{Turoyo~Aramaic}{TUA}
1898 \newfontlanguage{Tulu}{TUL}\newfontlanguage{Tuvini}{TUV}
1899 \newfontlanguage{Twí}{TWI}\newfontlanguage{Udmurt}{UDM}
1900 \newfontlanguage{Ukrainian}{UKR}\newfontlanguage{Urdu}{URD}
1901 \newfontlanguage{Upper~Sorbian}{USB}\newfontlanguage{Uyghur}{UYG}
1902 \newfontlanguage{Uzbek}{UZB}\newfontlanguage{Venda}{VEN}
1903 \newfontlanguage{Vietnamese}{VIT}\newfontlanguage{Wa}{WA}
1904 \newfontlanguage{Wagdi}{WAG}\newfontlanguage{West-Cree}{WCR}
1905 \newfontlanguage{Welsh}{WEL}\newfontlanguage{Wolof}{WLF}
1906 \newfontlanguage{Tai~Lue}{XBD}\newfontlanguage{Xhosa}{XHS}
1907 \newfontlanguage{Yakut}{YAK}\newfontlanguage{Yoruba}{YBA}
1908 \newfontlanguage{Y-Cree}{YCR}\newfontlanguage{Yi~Classic}{YIC}
1909 \newfontlanguage{Yi~Modern}{YIM}\newfontlanguage{Chinese~Hong~Kong}{ZHH}
1910 \newfontlanguage{Chinese~Phonetic}{ZHP}\newfontlanguage{Chinese~Simplified}{ZHS}
1911 \newfontlanguage{Chinese~Traditional}{ZHT}\newfontlanguage{Zande}{ZND}
1912 \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:

```

1913 \define@key{zf@feat}{Lang}{Turkish}[]{}
1914 \fontspec_check_lang:nTF {TRK} {
1915   \c@zf@language\l_fontspec_strnum_int\relax
1916   \zf@update@family{+lang=Turkish}
1917   \tl_set:Nn \l_fontspec_lang_tl {TRK}
1918 }{
1919   \fontspec_check_lang:nTF {TUR} {

```

```

1920 \c@zf@language\l_fontspec_strnum_int\relax
1921 \zf@update@family{+lang=Turkish}
1922 \tl_set:Nn \l_fontspec_lang_tl {TUR}
1923 }{
1924 \fontspec_warning:nx {language-not-exist} {#1}
1925 }
1926 }
1927 }

```

Default

```

1928 \define@key[zf@feat]{Lang}{Default}[]{}
1929 \zf@update@family{+lang=dflt}
1930 \tl_set:Nn \l_fontspec_lang_tl {DFLT}
1931 \c@zf@language=0\relax
1932 }

```

23.10.23 Raw feature string

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

```

1933 \define@key[zf]{options}{RawFeature}{
1934 \zf@update@family{+Raw:#1}
1935 \zf@update@ff{#1}
1936 }

```

23.11 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.

```

1937 \providecommand*\sidefault{}{si}
1938 \DeclareRobustCommand{\sishape}{
1939 \not@math@alphabet\sishape\relax
1940 \fontshape\sidefault\selectfont
1941 }
1942 \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.

```

1943 \newcommand*\zf@merge@shape[3]{
1944 \edef\@tempa{#1}
1945 \edef\@tempb{#2}
1946 \ifx\f@shape\@tempb
1947 \ifcsname\f@encoding/\f@family/\f@series/#3\endcsname

```

```

1948     \edef\@tempa{#3}
1949     \fi
1950     \fi
1951     \fontshape{\@tempa}\selectfont
1952 }

```

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

```

\scshape 1953 \DeclareRobustCommand \itshape {
\upshape 1954   \not@math@alphabet\itshape\mathit
1955   \zf@merge@shape\itdefault\scdefault\sidefault
1956 }
1957 \DeclareRobustCommand \slshape {
1958   \not@math@alphabet\slshape\relax
1959   \zf@merge@shape\sldefault\scdefault\sidefault
1960 }
1961 \DeclareRobustCommand \scshape {
1962   \not@math@alphabet\scshape\relax
1963   \zf@merge@shape\scdefault\itdefault\sidefault
1964 }
1965 \DeclareRobustCommand \upshape {
1966   \not@math@alphabet\upshape\relax
1967   \zf@merge@shape\updefault\sidefault\scdefault
1968 }

```

23.12 Selecting maths fonts

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

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

Since every maths setup is slightly different, we also take different paths for defining various math glyphs depending which maths font package has been loaded.

```

1969 \@ifpackageloaded{euler}{\zf@package@euler@loadedtrue}
1970                             {\zf@package@euler@loadedfalse}
1971 \def\zf@math{
1972   \let\zf@font@warning\@font@warning
1973   \let\@font@warning\@font@info
1974   \@ifpackageloaded{euler}{
1975     \ifzf@package@euler@loaded
1976       \zf@math@eulertrue
1977     \else
1978       \fontspec_error:n {euler-too-late}
1979     \fi
1980   }{}
1981   \@ifpackageloaded{lucbmath}{\zf@math@lucidatrue}{}

```

```

1982 \ifpackageloaded{lucidabr}{\zf@math@lucidatrue}{}
1983 \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.)

```

1984 \DeclareSymbolFont{legacymaths}{OT1}{cmr}{m}{n}
1985 \SetSymbolFont{legacymaths}{bold}{OT1}{cmr}{bx}{n}
1986 \DeclareMathAccent{\acute}{\mathalpha}{legacymaths}{19}
1987 \DeclareMathAccent{\grave}{\mathalpha}{legacymaths}{18}
1988 \DeclareMathAccent{\ddot}{\mathalpha}{legacymaths}{127}
1989 \DeclareMathAccent{\tilde}{\mathalpha}{legacymaths}{126}
1990 \DeclareMathAccent{\bar}{\mathalpha}{legacymaths}{22}
1991 \DeclareMathAccent{\breve}{\mathalpha}{legacymaths}{21}
1992 \DeclareMathAccent{\check}{\mathalpha}{legacymaths}{20}
1993 \DeclareMathAccent{\hat}{\mathalpha}{legacymaths}{94} % too bad, euler
1994 \DeclareMathAccent{\dot}{\mathalpha}{legacymaths}{95}
1995 \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 `amsmath` is loaded since we want to keep its definition.

```

1996 \begingroup
1997   \mathchardef\@tempa="603A \relax
1998   \let\next\egroup
1999   \ifx\colon\@tempa
2000     \DeclareMathSymbol{\colon}{\mathpunct}{legacymaths}{58}
2001   \fi
2002 \endgroup

```

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

```

2003 \ifzf@math@euler\else
2004   \DeclareMathSymbol{!}{\mathclose}{legacymaths}{33}
2005   \DeclareMathSymbol{:}{\mathrel}{legacymaths}{58}
2006   \DeclareMathSymbol{;}{\mathpunct}{legacymaths}{59}
2007   \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.

```

2008 \ifzf@math@lucida\else
2009   \DeclareMathSymbol{0}{\mathalpha}{legacymaths}{‘0}
2010   \DeclareMathSymbol{1}{\mathalpha}{legacymaths}{‘1}
2011   \DeclareMathSymbol{2}{\mathalpha}{legacymaths}{‘2}
2012   \DeclareMathSymbol{3}{\mathalpha}{legacymaths}{‘3}
2013   \DeclareMathSymbol{4}{\mathalpha}{legacymaths}{‘4}
2014   \DeclareMathSymbol{5}{\mathalpha}{legacymaths}{‘5}
2015   \DeclareMathSymbol{6}{\mathalpha}{legacymaths}{‘6}
2016   \DeclareMathSymbol{7}{\mathalpha}{legacymaths}{‘7}
2017   \DeclareMathSymbol{8}{\mathalpha}{legacymaths}{‘8}
2018   \DeclareMathSymbol{9}{\mathalpha}{legacymaths}{‘9}
2019   \DeclareMathSymbol{\Gamma}{\mathalpha}{legacymaths}{0}
2020   \DeclareMathSymbol{\Delta}{\mathalpha}{legacymaths}{1}
2021   \DeclareMathSymbol{\Theta}{\mathalpha}{legacymaths}{2}
2022   \DeclareMathSymbol{\Lambda}{\mathalpha}{legacymaths}{3}
2023   \DeclareMathSymbol{\Xi}{\mathalpha}{legacymaths}{4}
2024   \DeclareMathSymbol{\Pi}{\mathalpha}{legacymaths}{5}
2025   \DeclareMathSymbol{\Sigma}{\mathalpha}{legacymaths}{6}
2026   \DeclareMathSymbol{\Upsilon}{\mathalpha}{legacymaths}{7}
2027   \DeclareMathSymbol{\Phi}{\mathalpha}{legacymaths}{8}
2028   \DeclareMathSymbol{\Psi}{\mathalpha}{legacymaths}{9}
2029   \DeclareMathSymbol{\Omega}{\mathalpha}{legacymaths}{10}
2030   \DeclareMathSymbol{+}{\mathbin}{legacymaths}{43}
2031   \DeclareMathSymbol{=}{\mathrel}{legacymaths}{61}
2032   \DeclareMathDelimiter{(}{\mathopen}{legacymaths}{40}{largesymbols}{0}
2033   \DeclareMathDelimiter{)}{\mathclose}{legacymaths}{41}{largesymbols}{1}
2034   \DeclareMathDelimiter{[}{\mathopen}{legacymaths}{91}{largesymbols}{2}
2035   \DeclareMathDelimiter{]}{\mathclose}{legacymaths}{93}{largesymbols}{3}
2036   \DeclareMathDelimiter{/}{\mathord}{legacymaths}{47}{largesymbols}{14}
2037   \DeclareMathSymbol{\mathdollar}{\mathord}{legacymaths}{36}
2038 \fi
2039 \fi

```

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

with the `\setmathrm(...)` commands in the preamble.

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

```

2040 \DeclareSymbolFont{operators}\zf@enc\zf@rmmaths\mddefault\updefault
2041 \SetSymbolFont{operators}{normal}\zf@enc\zf@rmmaths\mddefault\updefault
2042 \SetMathAlphabet\mathrm{normal}\zf@enc\zf@rmmaths\mddefault\updefault
2043 \SetMathAlphabet\mathit{normal}\zf@enc\zf@rmmaths\mddefault\itdefault
2044 \SetMathAlphabet\mathbf{normal}\zf@enc\zf@rmmaths\bfdefault\updefault
2045 \SetMathAlphabet\mathsf{normal}\zf@enc\zf@sffmaths\mddefault\updefault
2046 \SetMathAlphabet\mathtt{normal}\zf@enc\zf@ttmaths\mddefault\updefault
2047 \SetSymbolFont{operators}{bold}\zf@enc\zf@rmmaths\bfdefault\updefault
2048 \ifdefined\zf@rmboldmaths
2049   \SetMathAlphabet\mathrm{bold}\zf@enc\zf@rmboldmaths\mddefault\updefault
2050   \SetMathAlphabet\mathbf{bold}\zf@enc\zf@rmboldmaths\bfdefault\updefault
2051   \SetMathAlphabet\mathit{bold}\zf@enc\zf@rmboldmaths\mddefault\itdefault
2052 \else
2053   \SetMathAlphabet\mathrm{bold}\zf@enc\zf@rmmaths\bfdefault\updefault
2054   \SetMathAlphabet\mathit{bold}\zf@enc\zf@rmmaths\bfdefault\itdefault
2055 \fi
2056 \SetMathAlphabet\mathsf{bold}\zf@enc\zf@sffmaths\bfdefault\updefault
2057 \SetMathAlphabet\mathtt{bold}\zf@enc\zf@ttmaths\bfdefault\updefault
2058 \let\font@warning\zf@font@warning

```

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

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

```

2059 \def\zf@math@maybe{
2060   \ifpackageloaded{anttor}{
2061     \ifx\define@antt@mathversions a\zf@mathfalse\fi{}
2062   }
2063   \ifpackageloaded{arev}{\zf@mathfalse{}}
2064   \ifpackageloaded{eulervm}{\zf@mathfalse{}}
2065   \ifpackageloaded{mathdesign}{\zf@mathfalse{}}
2066   \ifpackageloaded{concmath}{\zf@mathfalse{}}
2067   \ifpackageloaded{cmbright}{\zf@mathfalse{}}
2068   \ifpackageloaded{mathsf}{\zf@mathfalse{}}
2069   \ifpackageloaded{gfsartemis}{\zf@mathfalse{}}
2070   \ifpackageloaded{gfsneohellenic}{\zf@mathfalse{}}
2071   \ifpackageloaded{iwona}{
2072     \ifx\define@iwona@mathversions a\zf@mathfalse\fi{}
2073   }
2074   \ifpackageloaded{kpfonts}{\zf@mathfalse{}}
2075   \ifpackageloaded{kmath}{\zf@mathfalse{}}
2076   \ifpackageloaded{kurier}{
2077     \ifx\define@kurier@mathversions a\zf@mathfalse\fi{}
2078   }
2079   \ifpackageloaded{fouriernc}{\zf@mathfalse{}}
2080   \ifpackageloaded{fourier}{\zf@mathfalse{}}
2081   \ifpackageloaded{mathpazo}{\zf@mathfalse{}}
2082   \ifpackageloaded{mathptmx}{\zf@mathfalse{}}

```

```

2080 \ifpackageloaded{MinionPro}{\zf@mathfalse}{}
2081 \ifpackageloaded{unicode-math}{\zf@mathfalse}{}
2082 \ifpackageloaded{breqn}{\zf@mathfalse}{}
2083 \if@zf@math
2084     \fontspec_info:n {setup-math}
2085     \zf@math
2086 \fi
2087 }
2088 \AtBeginDocument{\zf@math@maybe}

```

23.13 Finishing up

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

```

2089 \if@zf@configfile
2090     \InputIfFileExists{fontspec.cfg}
2091         {\typeout{fontspec.cfg~ loaded.}}
2092         {\typeout{No~ fontspec.cfg~ file~ found;~ no~ configuration~ loaded.}}
2093 \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
```

23.14 Unicode footnote symbols

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

23.15 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
6 {
7 \tl_if_eq:xxTF \f@shape \sldefault \eminnershape \emshape
8 }
9 }
10 \DeclareTextFontCommand{\emph}{\em}
11 \let\emshape\itshape
12 \let\eminnershape\upshape
```

23.16 \-

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

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

23.17 Verbatims

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

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

```
21 \cs_new:Npn \fontspec_visible_space: {
22 \fontspec_glyph_if_exist:NnTF \font {"2423}
23 {\char"2423\relax}
24 {\fontspec_visible_space_fallback:}
25 }
```

\fontspec_visible_space:@fallback If the current font doesn't have U+2434: OPEN BOX, use Latin Modern Mono instead.

```
26 \cs_new:Npn \fontspec_visible_space_fallback: {
```

```

27 {
28   \usefont{EU1}{lmtt}{\f@series}{\f@shape}
29   \textvisiblespace
30 }
31 }

```

fontspec_print_visible_spaces: Helper macro to turn spaces (^^20) active and print visible space instead.

```

32 \group_begin:
33 \char_make_active:n{"20}%
34 \cs_gset:Npn\fontspec_print_visible_spaces:{%
35 \char_make_active:n{"20}%
36 \cs_set_eq:NN^20\fontspec_visible_space:%
37 }%
38 \group_end:

```

\verb Redefine \verb to use \fontspec_print_visible_spaces:.

```

\verb* 39 \def\verb{
40   \relax\ifmmode\hbox\else\leavevmode\null\fi
41   \bgroup
42   \verb@eol@error \let\do\@makeoother \dospecials
43   \verbatim@font\@noligs
44   \ifstar\@sverb\@verb
45 }
46 \def\@sverb{\fontspec_print_visible_spaces:\@sverb}

```

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

```

47 \AtBeginDocument{
48   \fontspec_patch_verbatim:
49   \fontspec_patch_moreverb:
50   \fontspec_patch_fancyvrb:
51   \fontspec_patch_listings:
52 }

```

verbatim* With the verbatim package.

```

53 \cs_set:Npn \fontspec_patch_verbatim: {
54   \@ifpackageloaded{verbatim}{
55     \cs_set:cpn {verbatim*} {
56       \begingroup \@verbatim \fontspec_print_visible_spaces: \verbatim@start
57     }
58   }{

```

This is for vanilla L^AT_EX.

```

59   \cs_set:cpn {verbatim*} {
60     \@verbatim \fontspec_print_visible_spaces: \@sxverbatim
61   }
62 }
63 }

```

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

```

64 \cs_set:Npn \fontspec_patch_moreverb: {
65   \@ifpackageloaded{moreverb}{
66     \cs_set:cpn {listingcont*} {

```

```

67     \cs_set:Npn \verbatim@processline {
68         \thelisting@line \global\advance\listing@line\c_one
69         \the\verbatim@line\par
70     }
71     \@verbatim \fontspec_print_visible_spaces: \verbatim@start
72 }
73 {}
74 }

```

listings and fancvrb make things nice and easy:

```

75 \cs_set:Npn \fontspec_patch_fancyvrb: {
76     \@ifpackageloaded{fancyvrb}{
77         \cs_set_eq:NN \FancyVerbSpace \fontspec_visible_space:
78     }{}
79 }

80 \cs_set:Npn \fontspec_patch_listings: {
81     \@ifpackageloaded{listings}{
82         \cs_set_eq:NN \lst@visiblespace \fontspec_visible_space:
83     }{}
84 }

```

Part X

fontspec.cfg

```
1
2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3 %%% FOR BACKWARDS COMPATIBILITY WITH PREVIOUS VERSIONS %%%
4
5 % Please note that most of the entries here from fontspec v1.x are
6 % no longer present. Please advise of any serious problems this causes.
7
8 \aliasfontfeatureoption{Ligatures}{Historic}{Historical}
9 \let\newfontinstance\newfontfamily
10
```