The fontspec package

Will Robertson and Khaled Hosny

2010/06/08 v2.0

Contents

1 History

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

More recently, LuaTeX is fast becoming the TeX engine of the day; it supports

Unicode encodings and OpenType fonts and opens up the internals of TEX via the Lua programming language. Hans Hagen's ConTEXt Mk. IV is a re-write of his powerful typesetting system, taking full advantage of LuaTEX's features including font support; a kernel of his work in this area has been extracted to be useful for other TEX macro systems as well, and this has enabled fontspec to be adapted for LATEX when run with the LuaTEX engine. Elie Roux and Khaled Hosny have been instrumental and invaluable with this development work.

2 Introduction

The fontspec package allows users of either XETEX or LuaTEX 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 XETEX should have little or no difficulty switching over to LuaTEX.

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

2.1 About this manual

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

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

3 Package loading and options

For basic use, no package options are required:

```
\usepackage{fontspec}
```

XeTeX users only Ross Moore's xunicode package is recommended for providing backwards compatibility with LaTeX's methods for accessing extra characters and accents (for example, `\%`, `\$`, `\textbullet`, `\"u`, and so on), plus many more Unicode characters. The xltxtra package adds a couple of general improvements to LaTeX under XeTeX; it also provides the `\XeTeX` macro to typeset the XeTeX logo.

LuaTeX users only In order to load fonts by their name rather than by their filename (*e.g.*, 'Latin Modern Roman' instead of 'ec-lmr10'), you may need to run the script `mkluatexfontdb`, which is distributed with the luaotfload package. Note that if you do not execute this script beforehand, the first time you attempt to typeset the process will pause for (up to) several minutes. (But only the first time.) Please see the luaotfload documentation for more information. Note that the xunicode package is not required as it has been incorporated directly into the Unicode font definitions (see the euenc package for more information).

babel *The babel package is not really supported!* Especially Vietnamese, Greek, and Hebrew at least might not work correctly, as far as I can tell. There's a better chance with Cyrillic and Latin-based languages, however—fontspec ensures at least that fonts should load correctly, but hyphenation and other matters aren't guaranteed. Under 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 it is incorrectly changing the maths font when it should be leaving well enough alone, apply the `[no-math]` package option to manually suppress its maths font.

3.2 Configuration

If you wish to customise any part of the fontspec interface (see later in this manual, Section 14 on page 39 and **??**), this should be done by creating your own `fontspec.cfg` file,[1] which will be automatically loaded if it is found by XeTeX. Either place it in the same folder as the main document for isolated cases, or in a location that XeTeX 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 tran-

[1] An example is distributed with the package.

script (`.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 ⟨*font feature*⟩=⟨*option*⟩ lists; these are described in later:

- For general font features, see Section 8 on page 14
- For OpenType fonts, see Part II on page 18
- For XeTeX-only general font features, see Part III on page 28
- For features for AAT fonts in XeTeX, see Part Section 12 on page 31

4 Font selection

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

4.1 By font name

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

The simplest example might be something like

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

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

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

4.2 By file name

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

Note that XeTeX with the `xdvipdfmx` driver and LuaTeX can both select fonts in this way, but XeTeX with the `xdv2pdf` driver can only select fonts by name and not by file name. The `xdvipdfmx` driver is default for XeTeX; the `xdv2pdf` driver is only available on Mac OS X.

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

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

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

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

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

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

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

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

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

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

```
\setmainfont{TeX Gyre Bonum}
\setsansfont[Scale=MatchLowercase]{Latin Modern Sans}
\setmonofont[Scale=MatchLowercase]{Inconsolata}
\rmfamily\pangram\par
\sffamily\pangram\par
\ttfamily\pangram
```

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

Note that XeTeX and LuaTeX are able to load the font without giving an extension, but fontspec must know to search for the file; this can can be indicated by declaring the font exists in an 'ExternalLocation':

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

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

5 Default font families

\setmainfont [⟨*font features*⟩] {⟨*font name*⟩}
\setsansfont [⟨*font features*⟩] {⟨*font name*⟩}
\setmonofont [⟨*font features*⟩] {⟨*font name*⟩}

These commands are used to select the default font families for the entire document. They take the same arguments as \fontspec. See Example 1. Here, the scales of the fonts have been chosen to equalise their lowercase letter heights. The Scale font feature will be discussed further in Section 8 on page 14, including methods for automatic scaling.

6 New commands to select font families

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

For cases when a specific font with a specific feature set is going to be re-used many times in a document, it is inefficient to keep calling \fontspec for every use. While the command does not define a new font instance after the first call, the feature options must still be parsed and processed.

\newfontfamily For this reason, new commands can be created for loading a particular font family with the \newfontfamily command, demonstrated in Example 2. This

Example 2: Defining new font families.

This is a *note*.

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

Example 3: Defining a single font face.

where is all the vegemite

```
\newfontface\fancy
    [Contextuals={WordInitial,WordFinal}]
                        {Hoefler Text Italic}
\fancy where is all the vegemite
% \emph, \textbf, etc., all don't work
```

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

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

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

`\newfontface` Sometimes only a specific font face is desired, without accompanying italic or bold variants begin automatically selected. This is common when selecting a fancy italic font, say, that has swash features unavailable in the upright forms. `\newfontface` is used for this purpose, shown in Example 3, which is repeated in Section 12.4 on page 32.

6.1 More control over font shape selection

`BoldFont` = ⟨*font name*⟩
`ItalicFont` = ⟨*font name*⟩
`BoldItalicFont` = ⟨*font name*⟩
`SlantedFont` = ⟨*font name*⟩
`BoldSlantedFont` = ⟨*font name*⟩
`SmallCapsFont` = ⟨*font name*⟩

The automatic bold, italic, and bold italic font selections will not be adequate for the needs of every font: while some fonts mayn't even have bold or italic shapes, in which case a skilled (or lucky) designer may be able to chose well-matching accompanying shapes from a different font altogether, others can have a range of bold and italic fonts to chose between. The `BoldFont` and `ItalicFont` 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.

Example 4: Explicit selection of the bold font.

Helvetica Neue UltraLight
Helvetica Neue UltraLight Italic
Helvetica Neue
Helvetica Neue Italic

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

6.1.1 Input shorthands

For those cases that the base font name is repeated, you can replace it with an asterisk. (This has been shown previously in this section.) For example, some space can be saved instead of writing 'Baskerville SemiBold':

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

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

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

6.1.2 Small caps and slanted font shapes

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

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

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

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

6.2 Math(s) fonts

When `\setmainfont`, `\setsansfont` and `\setmonofont` are used in the preamble, they also define the fonts to be used in maths mode inside the `\mathrm`-type commands. This only occurs in the preamble because LaTeX freezes the maths fonts after this stage of the processing. The fontspec package must also be loaded after

any maths font packages (*e.g.*, euler) to be successful. (Actually, it is *only* euler that is the problem.[2])

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

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

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

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

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

6.3 Miscellaneous font selecting details

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

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

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

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

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

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

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

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

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

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

Some ‘default’ Didot 0123456789
Now grey, with old-style gures: 0123456789

7 Selecting font features

The commands discussed so far each take an optional argument for accessing the font features of the requested font. These features are generally unavailable or harder to access in regular LaTeX.

7.1 Default settings

`\defaultfontfeatures{`⟨*font features*⟩`}`

It is desirable to define options that are applied to every subsequent font selection command: a default feature set, so to speak. This may be defined with the `\defaultfontfeatures` command, shown in Example 5. New calls of `\defaultfontfeatures` overwrite previous ones.

7.2 Changing the currently selected features

`\addfontfeatures{`⟨*font features*⟩`}`

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

‘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

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

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

Example 7: Features for, say, just italics.

Attention All Martini Drinkers
Attention All Martini Drinkers

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

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`

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

Skia
Skia 'Bold'

```
\fontspec[BoldFont={Skia},
 BoldFeatures={Weight=2}]{Skia}
Skia \\ \bfseries Skia 'Bold'
```

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

Upright Small Caps
Italic Italic Small Caps
Bold Bold Small Caps
Bold Italic Bold Italic Small Caps

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

can be nested within `ItalicFeatures` and friends. Every combination of upright, italic, bold and small caps can thus be assigned individual features, as shown in the somewhat ludicrous Example 9.

7.5 Different features for different font sizes

```
SizeFeatures = {
  ...
  { Size = ⟨size range⟩, ⟨font features⟩ }
  { Size = ⟨size range⟩, Font = ⟨font name⟩, ⟨font features⟩ }
  ...
}
```

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

It takes a comma separated list of braced, comma separated lists of features for each size range. Each sub-list must contain the `Size` option to declare the size range, and optionally `Font` to change the font based on size. Other (regular) fontspec features that are added are used on top of the font features that would 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 11.5 on page 31.

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

```
\fontspec[ SizeFeatures={
    {Size={-8}, Font=TeX Gyre Bonum Italic, Color=AA0000},
    {Size={8-14}, Color=00AA00},
    {Size={14-}, Color=0000AA}} ]{TeX Gyre Chorus}

{\scriptsize Small\par} Normal size\par {\Large Large\par}
```

Small
Normal size
Large

Input	Font size, s
`Size = X-`	$s \geqslant X$
`Size = -Y`	$s < Y$
`Size = X-Y`	$X \leqslant s < Y$
`Size = X`	$s = X$

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

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

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

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

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

8 Font independent options

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

8.1 Color

`Color` (or `Colour`), also shown in Section 7.1 on page 11 and elsewhere, uses font specifications to set the color of the text. The color is defined as a triplet of two-digit Hex RGB values, with optionally another value for the transparency (where `00` is completely transparent and `FF` is opaque.) Transparency is supported by XeLaTeX with the `xdv2pdf` driver and with LuaLaTeX; XeLaTeX with the `xdvipdfmx` driver is not supported.

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

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}
```

Example 12: Automatically calculated scale values.

The perfect match is hard to find.
L O G O F O N T

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

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

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

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

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

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

8.2 Scale

`Scale` = ⟨*number*⟩
`Scale` = `MatchLowercase`
`Scale` = `MatchUppercase`

In its explicit form, `Scale` takes a single numeric argument for linearly scaling the font, as demonstrated in Section 5 on page 7. It is now possible to measure the correct dimensions of the fonts loaded and calculate values to scale them automatically.

As well as a numerical argument, `Scale` feature also accepts options `MatchLowercase` and `MatchUppercase`, which will scale the font being selected to match the current default roman font to either the height of the lowercase or uppercase letters, respectively; these features are shown in Example 12.

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

```
\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.
```

Some text for our example to take up some space, and to demonstrate the default interword space.

Some text for our example to take up some space, and to demonstrate the default interword space.

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

8.4 Post-punctuation space

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

The `PunctuationSpace` feature takes a scaling factor by which to adjust the nominal value chosen for the font; this is demonstrated in Example 14. Note that `PunctuationSpace=0` is *not* equivalent to `\frenchspacing`, although the difference will only be apparent when a line of text is under-full.

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

Example 14: Scaling the default post-punctuation space.

Letters, Words. Sentences.
Letters, Words. Sentences.
Letters, Words. Sentences.

```
\nonfrenchspacing
\fontspec{TeX Gyre Schola}
 Letters, Words. Sentences.          \par
\fontspec[PunctuationSpace=2]{TeX Gyre Schola}
 Letters, Words. Sentences.          \par
\fontspec[PunctuationSpace=0]{TeX Gyre Schola}
 Letters, Words. Sentences.
```

Example 15: Explicitly choosing the hyphenation character.

EXAMPLE HYPHENATION

EXAMPLE HY+ PHEN+ ATION

EXAMPLE HYPHEN- ATION

```
\def\text{\fbox{\parbox{1.55cm}{%
  EXAMPLE HYPHENATION%
}}\qquad\qquad\null\par\bigskip}

\fontspec{Adobe Garamond Pro}
\addfontfeature{HyphenChar=None}
\text
\addfontfeature{HyphenChar={+}}
\text
\addfontfeature{HyphenChar={"F6BA}}
\text
```

the input is the string `None`, then hyphenation is suppressed for this font.[3] 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.

Adobe Garamond Pro's uppercase hyphenation character is used to demonstrate a possible use for this feature in Example 15. Note that in an actual situation, the `Uppercase` option of the `Letters` feature would probably supply this for you (see Section 12.2 on page 32).

This package redefines LaTeX's `\-` macro such that it adjusts along with the above changes.

8.6 Optical font sizes

Optically scaled fonts thicken out as the font size decreases in order to make the glyph shapes more robust (less prone to losing detail), which improves legibility. Conversely, at large optical sizes the serifs and other small details may be more delicately rendered.

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

[3] Known bug: this doesn't work yet in LuaLaTeX.

Example 16: Optical size substitution is suppressed when set to zero.

Warnock Pro optical sizes
Warnock Pro optical sizes
Warnock Pro optical sizes
Warnock Pro optical sizes

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

Example 17: A demonstration of automatic optical size selection.

Automatic optical size
Automatic optical size

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

For the OpenType font Warnock Pro, we have three optically sized variants: caption, subhead, and display. With `OpticalSize` set to zero, no optical size font substitution is performed, as shown in Example 16.

Automatic OpenType optical scaling is in Example 17, in which we've scaled down some large text in order to be able to compare the difference for equivalent font sizes (this gives the same output as we saw in the previous example for Warnock Pro Display).

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

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

Part II
OpenType

9 Introduction

TODO: explain OpenType font features.

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

Feature		Option		Tag	
Ligatures	=	Required	∗	`rlig`	
		NoRequired		`rlig`	(*deactivate*)
		Common	∗	`liga`	
		NoCommon		`liga`	(*deactivate*)
		Contextual	∗	`clig`	
		NoContextual		`clig`	(*deactivate*)
		Rare/Discretionary		`dlig`	
		Historical		`hlig`	
		TeX		`tlig/trep`	

∗ This feature is activated by default.

Table 2: Options for the OpenType font feature 'Ligatures'.

Example 18: An example of the `Ligatures` feature.

strict firefly
strict firefly

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

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

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

10 Complete listing of OpenType font features

10.1 Ligatures

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

10.2 Letters

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

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

Table 3: Options for the OpenType font feature 'Letters'.

Example 19: Small caps from lowercase or uppercase letters.

THIS SENTENCE NO VERB
THIS SENTENCE NO VERB

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

Petite caps are smaller than small caps. `SmallCaps` and `PetiteCaps` turn lowercase letters into the smaller caps letters, whereas the `Uppercase...` options turn the *capital* letters into the smaller caps (good, *e.g.*, for applying to already uppercase acronyms like 'NASA'). This difference is shown in Example 19. 'Unicase' is a weird hybrid of upper and lower case letters.

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

The `Kerning` feature also contains an `Uppercase` option, which adds a small amount of spacing in between letters (see Section 10.11 on page 24).

10.3 Numbers

The `Numbers` feature defines how numbers will look in the selected font, accepting options shown in Table 4.

The synonyms `Uppercase` and `Lowercase` are equivalent to `Lining` and `OldStyle`, respectively. The differences have been shown previously in Section 7.2 on page 11.

[4] If you want automatic uppercase letters, look to LaTeX's `\MakeUppercase` command.

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

UPPER-CASE example
UPPER-CASE example

```
\fontspec{Adobe Garamond Pro}
 UPPER-CASE example \\
\addfontfeature{Letters=Uppercase}
 UPPER-CASE example
```

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

Table 4: Options for the OpenType font feature 'Numbers'.

Example 21: The effect of the SlashedZero option.

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

0123456789 0123456789

The Monospaced option is useful for tabular material when digits need to be vertically aligned.

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

The Remap option maps numerals to their Arabic or Farsi equivalents based on the current Language setting (see Section 10.15 on page 27). This is based on a LuaTEX feature of the luaotfload package, not an OpenType feature.

10.4 Contextuals

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

Historic forms are accessed in OpenType fonts via the feature Style=Historic; this is generally *not* contextual in OpenType, which is why it is not included here.

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

Table 5: Options for the OpenType font feature 'Contextuals'.

Example 22: An example of the Swashes option of the Contextuals feature.

Without Contextual Swashes
With Contextual Swashes; cf. W C S

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

Feature	Option	Tag
VerticalPosition =	Superior	sups
	Inferior	subs
	Numerator	numr
	Denominator	dnom
	ScientificInferior	sinf
	Ordinal	ordn

Table 6: Options for the OpenType font feature 'VerticalPosition'.

10.5 Vertical Position

The VerticalPosition feature is used to access things like subscript (Inferior) and superscript (Superior) numbers and letters (and a small amount of punctuation, sometimes). The Ordinal option will only raise characters that are used in some languages directly after a number. The ScientificInferior feature will move glyphs further below the baseline than the Inferior feature. These are all shown in Example 23

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

The realscripts package (yet to be released) (or xltxtra for XeTeX) redefines the \textsubscript and \textsuperscript commands to use the above font features automatically.

Example 23: The VerticalPosition feature.

Sup: abdehilmnorst (-$12,345.67)
Numerator: 12345
Denominator: 12345
Scientific Inferior: 12345
'Ordinals': 1st 2nd 3rd 4th 0th

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

Feature	Option	Tag
Fractions =	On	frac
	Alternate	afrc

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

Example 24: The Fractions feature.

1/2 1/4 5/6 13579/24680
½ ¼ ⅚ 13579/24680
½ ¼ ⅚ 13579/24680

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

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 24.

10.7 StylisticSet

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

Two demonstrations from the Junicode font[5] are shown in Example 25 and Example 26; thanks to Adam Buchbinder for the suggestion.

(This is a synonym of the `Variant` feature for AAT fonts.) See Section 14 on page 39 for a way to assign names to stylistic sets, which should be done on a per-font basis.

10.8 Alternates

Selection of the `Alternate` feature again must be done numerically.

[5] http://junicode.sf.net

Example 25: Insular (ancient) letterforms for the Junicode font, accessed with the `StylisticSet` feature.

Insular forms.
Inſulaꝛ ꝓoꝛmſ.

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

Example 26: Enlarged minuscules (capital letters remain unchanged) for the Junicode font, accessed with the `STylisticSet` feature.

ENLARGED Minuscules.
ENLARGED Minuscules.

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

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

Table 8: Options for the OpenType font feature 'Style'.

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

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

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

See Section 14 on page 39 for a way to assign names to alternates, which should be done on a per-font basis.

10.9 Style

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

Typical examples for these features are shown in Section 12.9.

10.10 Diacritics

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

10.11 Kerning

Specifies how inter-glyph spacing should behave.

Feature		Option		Tag
Diacritics	=	MarkToBase	∗	`mark`
		NoMarkToBase		`mark` (*deactivate*)
		MarkToMark	∗	`mkmk`
		NoMarkToMark		`mkmk` (*deactivate*)
		AboveBase	∗	`abvm`
		NoAboveBase		`abvm` (*deactivate*)
		BelowBase	∗	`blwm`
		NoBelowBase		`blwm` (*deactivate*)

∗ This feature is activated by default.

Table 9: Options for the OpenType font feature 'Diacritics'.

Feature		Option		Tag
Kerning	=	Uppercase		`cpsp`
		On	∗	`kern`
		Off		`kern` (*deactivate*)

∗ This feature is activated by default.

Table 10: Options for the OpenType font feature 'Kerning'.

As briefly mentioned previously at the end of Section 12.2 on page 32, the `Uppercase` option will add a small amount of tracking between uppercase letters, seen in Example 27.

10.12 CJK shape

There have been many standards for how CJK ideographic glyphs are 'supposed' to look. Some fonts will contain many alternate glyphs available in order to be able to display these gylphs 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.

Example 27: Adding extra kerning for uppercase letters.

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

UPPER-CASE EXAMPLE
UPPER-CASE EXAMPLE

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

Table 11: Options for the OpenType font feature 'CJKShape'.

Example 28: Different standards for CJK ideograph presentation.

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

唖噛躯 妍并訝
唖噛躯 妍幷訝
啞嚙軀 妍并訝

10.13 Character width

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

Japanese alphabetic glyphs (in Hiragana or Katakana) may be typeset proportionally, to better fit horizontal measures, or monospaced, to fit into the rigid grid imposed by ideographic typesetting. In this latter case, there are also half-width forms for squeezing more kana glyphs (which are less complex than the kanji they are amongst) into a given block of space. The same features are given to roman letters in Japanese fonts, for typesetting foreign words in the same style as the surrounding text.

The same situation occurs with numbers, which are provided in increasingly

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

Table 12: Options for the OpenType font feature 'CharacterWidth'.

Example 29: Proportional or fixed width forms.

ようこそ　ワカヨタレソ　abcdef
ようこそ　ワカヨタレソ　a b c d e f
ようこそ　ワカヨタレソ　abcdef

```
\def\test{\makebox[2cm][l]{\texta}%
          \makebox[2.5cm][l]{\textb}%
          \makebox[2.5cm][l]{abcdef}}
\fontspec{Hiragino Mincho Pro}
{\addfontfeature{CharacterWidth=Proportional}\test}\\
{\addfontfeature{CharacterWidth=Full}\test}\\
{\addfontfeature{CharacterWidth=Half}\test}
```

Example 30: Numbers can be compressed significantly.

—１２３２１—
-1234554321-
-123456787654321-
-12345678900987654321-

```
\fontspec[Renderer=AAT]{Hiragino Mincho Pro}
{\addfontfeature{CharacterWidth=Full}
 ---12321---}\\
{\addfontfeature{CharacterWidth=Half}
 ---1234554321---}\\
{\addfontfeature{CharacterWidth=Third}
 ---123456787654321---}\\
{\addfontfeature{CharacterWidth=Quarter}
 ---12345678900987654321---}
```

illegible compressed forms seen in exrefcharwd.

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

10.14 Vertical typesetting

TODO!

10.15 OpenType scripts and languages

When dealing with fonts that include glyphs for various languages, they may contain different font features for the different character sets and languages it supports. These may be selected with the `Script` and `Language` features. The possible options are tabulated in Table 13 on the following page and Table 14 on page 29, respectively. When a script or language is requested that is not supported by the current font, a warning is printed in the console output.

Because these font features can change which features are able to be selected for the font, they are selected by fontspec before all others and will specifically select the `ICU` renderer for this font, as described in Section 11.4 on page 31.

10.15.1 Defining new scripts and languages

`\newfontscript` `\newfontlanguage`

Further scripts and languages may be added with the `\newfontscript` and `\newfontlanguage` commands. For example,

Example 31: XeTeX's `Mapping` feature.

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

```
\newfontscript{Arabic}{arab}
\newfontlanguage{Turkish}{TUR}
```

The first argument is the fontspec name, the second the OpenType definition. The advantage to using these commands rather than `\newfontfeature` (see Section 14 on page 39) is the error-checking that is performed when the script or language is requested.

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

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

Part III
Fonts and features with XeTeX

11 XeTeX-only font features

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

11.1 Mapping

`Mapping` enables a XeTeX text-mapping scheme, shown in Example 31.

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

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

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

Example 32: 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
```

Example 33: 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
```

11.2 Letter spacing

Letter spacing, or tracking, is the term given to adding (or subtracting) a small amount of horizontal space in between adjacent characters. It is specified with the LetterSpace, which takes a numeric argument, shown in Example 32.

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 12.2 on page 32).

11.3 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 33. 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.

11.4 Different font technologies: AAT and ICU

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

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

Among some other font features only available through a specific renderer, `ICU` provides for the `Script` and `Language` features, which allow different font behaviour for different alphabets and languages; see Section 10.15 on page 27 for the description of these features. *Because these font features can change which features are able to be selected for the font instance, they are selected by* fontspec *before all others and will automatically and without warning select the* `ICU` *renderer.*

11.5 Optical font sizes

Multiple Master fonts are parameterised over orthogonal font axes, allowing continuous selection along such features as weight, width, and optical size (see Section 13 on page 38 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 LaTeX, and the optical size for a Multiple Master font must always be specified explicitly.

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

12 Mac OS X's AAT fonts

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

Nonetheless, this is the font format that was first supported by XeTeX (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 XeLaTeX using the `xdv2pdf` driver, as opposed to the default `xdvipdfmx`. E.g.,

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

Mac OS X also supports Multiple Master fonts, which are discussed in Section 13.

12.1 Ligatures

`Ligatures` refer to the replacement of two separate characters with a specially drawn glyph for functional or æsthetic reasons. For AAT fonts, you may choose from any combination of `Required`, `Common`, `Rare` (or `Discretionary`), `Logos`, `Rebus`, `Diphthong`, `Squared`, `AbbrevSquared`, and `Icelandic`.

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

12.2 Letters

The `Letters` feature specifies how the letters in the current font will look. For AAT fonts, you may choose from `Normal`, `Uppercase`, `Lowercase`, `SmallCaps`, and `InitialCaps`.

12.3 Numbers

The `Numbers` feature defines how numbers will look in the selected font. For both AAT, they may be a combination of `Lining` or `OldStyle` and `Proportional` or `Monospaced` (the latter is good for tabular material). The synonyms `Uppercase` and `Lowercase` are equivalent to `Lining` and `OldStyle`, respectively. The differences have been shown previously in Section 7.2 on page 11.

12.4 Contextuals

This feature refers to glyph substitution that vary by their position; things like contextual swashes are implemented here The options for AAT fonts are `WordInitial`, `WordFinal` (Example 34), `LineInitial`, `LineFinal`, and `Inner` (Example **??**, also called 'non-final' sometimes). As non-exclusive selectors, like the ligatures, you can turn them off by prefixing their name with `No`.

12.5 Vertical position

The `VerticalPosition` feature is used to access things like subscript (`Inferior`) and superscript (`Superior`) numbers and letters (and a small amount of punctuation, sometimes). The `Ordinal` option is (supposed to be) contextually sensitive to only raise characters that appear directly after a number. These are shown in Example 36.

Example 34: Contextual glyph for the beginnings and ends of words.

where is all the vegemite

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

Example 35: A contextual feature for the 'long s' can be convenient as the character does not need to be marked up explicitly.

'Inner' ſwaſhes can *ſometimes* contain the archaic long s.

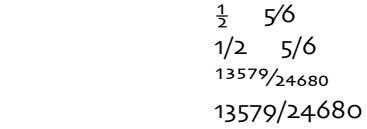
```
\fontspec[Contextuals=Inner]{Hoefler Text}
'Inner' swashes can \emph{sometimes}    \\
 contain the archaic long~s.
```

Example 36: Vertical position for AAT fonts.

Normal $^{\text{superior}}$ $_{\text{inferior}}$
1^{st} 2^{nd} 3^{rd} 4^{th} 0^{th} 8^{abcde}

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

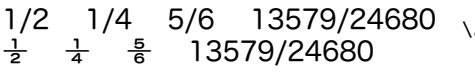
Example 37: Fractions in AAT fonts.




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

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

Example 38: Alternate design of pre-composed fractions.




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

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

12.6 Fractions

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

In AAT fonts, the 'fraction slash' or solidus character, is to be used to create fractions. When `Fractions` are turned `On`, then only pre-drawn fractions will be used. See Example 37.

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 38.

12.7 Variants

The `Variant` feature takes a single numerical input for choosing different alphabetic shapes. Don't mind my fancy Example 39 `:)` I'm just looping through the nine (`!`) variants of Zapfino.

See Section 14 on page 39 for a way to assign names to variants, which should be done on a per-font basis.

12.8 Alternates

Selection of `Alternates` *again* must be done numerically; see Example 40. See Section 14 on page 39 for a way to assign names to alternates, which should be done on a per-font basis.

Example 39: 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 40: Alternate shape selection must be numerical.

Sphinx Of Black Quartz, Judge My Vow
Sphinx Of Black Quartz, Judge My Vow

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

12.9 Style

The options of the `Style` feature are defined in AAT as one of the following: `Display`, `Engraved`, `IlluminatedCaps`, `Italic`, `Ruby`,[6] `TallCaps`, or `TitlingCaps`.

Example 41 and Example 42 both contain glyph subtitutions 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 43 and Example 44; in the latter, the `Italic` option affects the Latin text and the `Ruby` option the Japanese.

Note the difference here between the default and the horizontal style kana in Example 45.

12.10 CJK shape

There have been many standards for how CJK ideographic glyphs are 'supposed' to look. Some fonts will contain many alternate glyphs available in order to be

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

Example 41: Example of the `Alternate` option of the `Style` feature.

K Q R k v w y
K Q R k v w y

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

Example 42: Example of the `Historic` option of the `Style` feature.

M Q Z
M Q Z

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

Example 43: Example of the `TitlingCaps` option of the `Style` feature.

TITLING CAPS
TITLING CAPS

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

Example 44: Example of the `Italic` and `Ruby` options of the `Style` feature.

Latin ようこそ ワカヨタレソ
Latin ようこそ ワカヨタレソ

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

Example 45: Example of the `HorizontalKana` and `VerticalKana` options of the `Style` feature.

ようこそ ワカヨタレソ
ようこそ ワカヨタレソ
ようこそ ワカヨタレソ

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

Example 46: Annotation forms.

1 2 3 4 5 6 7 8 9
(1) (2) (3) (4) (5) (6) (7) (8) (9)
(1 (2 (3 (4 (5 (6 (7 (8 (9
1) 2) 3) 4) 5) 6) 7) 8) 9)
① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨
❶ ❷ ❸ ❹ ❺ ❻ ❼ ❽ ❾
🄀 🄁 🄂 🄃 🄄 🄅 🄆 🄇 🄈
🅐 🅑 🅒 🅓 🅔 🅕 🅖 🅗 🅘
🅰 🅱 🅲 🅳 🅴 🅵 🅶 🅷 🅸
🇦 🇧 🇨 🇩 🇪 🇫 🇬 🇭 🇮
🆐 🆑 🆒 🆓 🆔 🆕 🆖 🆗 🆘
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\x7\x8\x9
```

able to display these gylphs correctly in whichever form is appropriate. Both AAT and OpenType fonts support the following `CJKShape` options: `Traditional`, `Simplified`, `JIS1978`, `JIS1983`, `JIS1990`, and `Expert`. OpenType also supports the `NLC` option.

12.11 Character width

See Section 10.13 on page 26 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.

12.12 Annotation

Various Asian fonts are equipped with a more extensive range of numbers and numerals in different forms. For OpenType fonts, annotations with the `Annotation` feature are selected numerically, as shown in Example 46.

12.13 Vertical typesetting

TODO: improve!

XeTeX provides for vertical typesetting simply with the ability to rotate the individual glyphs as a font is used for typesetting, as shown in Example 47.

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

12.14 Diacritics

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

Example 47: Vertical typesetting.

共産主義者は

共産主義者よ

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

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

Example 48: Various annotation forms.

1 2 3 4 5 6 7 8 9
① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨
⑴ ⑵ ⑶ ⑷ ⑸ ⑹ ⑺ ⑻ ⑼
1. 2. 3. 4. 5. 6. 7. 8. 9.

```
\fontspec{Hei Regular}
 1 2 3 4 5 6 7 8 9                          \\
\fontspec[Annotation=Circle]{Hei Regular}
 1 2 3 4 5 6 7 8 9                          \\
\fontspec[Annotation=Parenthesis]{Hei Regular}
 1 2 3 4 5 6 7 8 9                          \\
\fontspec[Annotation=Period]{Hei Regular}
 1 2 3 4 5 6 7 8 9
```

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

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

12.15 Annotation

Various Asian fonts are equipped with a more extensive range of numbers and numerals in different forms. These are accessed through the `Annotation` feature with the following options: `Off`, `Box`, `RoundedBox`, `Circle`, `BlackCircle`, `Parenthesis`, `Period`, `RomanNumerals`, `Diamond`, `BlackSquare`, `BlackRoundSquare`, and `DoubleCircle`. See Example 48

13 AAT & Multiple Master font axes

Multiple Master and AAT font specifications both provide continuous variation along font parameters. For example, they don't have just regular and bold weights, they can have any bold weight you like between the two extremes.

`Weight`, `Width`, and `OpticalSize` are supported by this package. Skia, which is distributed with Mac OS X, has two of these variable parameters, allowing for the

Example 49: Continuously variable font parameters. These fonts are unfortunately quite rare.

Really light and extended Skia
Really fat and condensed Skia

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

Example 50: Assigning new AAT features.

This is XeTeX by Jonathan Kew.

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

demonstration in Example 49. Variations along a multiple master font's optical size axis has been shown previously in Section 11.5 on page 31.

Part IV
Programming interface

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

14 Defining new features

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

`\newAATfeature` New AAT features may be created with this command:

`\newAATfeature{`⟨*feature*⟩`}{`⟨*option*⟩`}{`⟨*feature code*⟩`}{`⟨*selector code*⟩`}`

Use the XeTeX file `AAT-info.tex` to obtain the code numbers. See Example 50.

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

`\newopentypefeature` `\newICUfeature{`⟨*feature*⟩`}{`⟨*option*⟩`}{`⟨*feature tag*⟩`}`

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

Here's what it would look like in practise:

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

\DescribeMacro{\newfontfeature}
In case the above commands do not accommodate the desired font feature
(perhaps a new \XeTeX\ feature that \pkg{fontspec} hasn't been updated
```

```
to support), a command is provided to pass arbitrary input into the
font selection string:\par
{\centering\cmd{\newfontfeature}\marg{name}\marg{input string}\par}

For example, Zapfino
contains the feature ‘Avoid d-collisions’. To access it
with this package, you could do some like that shown in \exref{avoidd}

\begin{Xexample}{avoidd}{Assigning new arbitary features.}
  \newfontfeature{AvoidD}{Special=Avoid d-collisions}
  \newfontfeature{NoAvoidD}{Special=!Avoid d-collisions}
  \fontspec[AvoidD,Variant=1]{Zapfino}
   sockdolager rubdown               \\
  \fontspec[NoAvoidD,Variant=1]{Zapfino}
   sockdolager rubdown
\end{Xexample}

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

\section{Going behind \pkg{fontspec}’s back}
Expert users may wish not to use \pkg{fontspec}’s feature handling at all,
while still taking advantage of its \LaTeX\ font selection conveniences. The
\feat{RawFeature} font feature allows literal \XeTeX\ font feature selection
when you happen to have the OpenType feature tag memorised.

\begin{Xexample}{raw}{Using raw font features directly.}
  \fontspec[RawFeature=+smcp]{TeX Gyre Pagella}
  Pagella small caps
\end{Xexample}

Multiple features can either be included in a single declaration:\par
{\centering|[RawFeature=+smcp;+onum]|\par}
\noindent or with multiple declarations:\par
{\centering|[RawFeature=+smcp, RawFeature=+onum]|\par}

\section{Renaming existing features \& options}
\label{sec:aliasfontfeature}

\DescribeMacro{\aliasfontfeature}
If you don’t like the name of a particular font feature,
it may be aliased to another with the
\cs{aliasfontfeature}\marg{existing name}\marg{new name} command,
such as shown in \exref{alias}.
```

```
\begin{Xexample}{alias}{Renaming font features.}
  \aliasfontfeature{ItalicFeatures}{IF}
  \fontspec[IF = {Alternate=1}]{Hoefler Text}
  Roman Letters \itshape And Swash
\end{Xexample}

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

\DescribeMacro{\aliasfontfeatureoption}
If you wish to change the name of a font feature option,
it can be aliased to another with the command
\cs{aliasfontfeatureoption}\marg{font feature}\marg{existing name}\marg{new name}, such as

\begin{Lexample}{aliasopt}{Renaming font feature options.}
  \aliasfontfeature{VerticalPosition}{Vert Pos}
  \aliasfontfeatureoption{VerticalPosition}{ScientificInferior}{Sci Inf}
  \fontspec[Vert Pos=Sci Inf]{Warnock Pro}
   Scientific Inferior: 12345
\end{Lexample}

This example demonstrates an important point: when aliasing the feature
options, the \emph{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, \opt{Proportional} can be aliased to \opt{Prop} in the
\feat{Letters} feature, but \opt{550099BB} cannot be substituted for \opt{Purple}
in a \feat{Color} specification. For this type of thing, the \cmd\newfontfeature\
command should be used to declare a new, \eg, \feat{PurpleColor} feature:
\begin{Verbatim}
  \newfontfeature{PurpleColor}{color=550099BB}
```

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

15 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`.

`\fontspec_if_fontspec_font:TF` Test whether the currently selected font has been loaded by fontspec.

`\fontspec_if_aat_feature:nnTF` Test whether the currently selected font contains the AAT feature (#1,#2).

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

`\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.

`\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.

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

`\fontspec_if_language: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.

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

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

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

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

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

Part V
The patching/improvement of LaTeX 2ε and other packages

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

16 Inner emphasis

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

17 Unicode footnote symbols

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

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

18 Verbatim

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

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

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

19 Discrectionary hyphenation: \-

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

Part VI
fontspec.sty

20 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*.)

```
\RequirePackage{expl3,xparse}
\input binhex.tex  % before expl syntax!
\ExplSyntaxOn
\msg_new:nnn {fontspec} {not-pdftex}
{
  Requires~ XeTeX~ or~ LuaTeX~ to~ function!
}
\xetex_if_engine:F {
  \luatex_if_engine:TF {
    \RequirePackage{luatextra}[2010/05/10]
    \luatexRequireModule{fontspec}
  }{
    \msg_error:nn {fontspec} {not-pdftex}
  }
}
```

`\xetex_or_luatex:nn` Use #1 if XeTeX or #2 if LuaTeX.

```
\xetex_if_engine:TF
  { \cs_new_eq:NN \xetex_or_luatex:nn \use_i:nn }
  { \luatex_if_engine:T
    { \cs_new_eq:NN \xetex_or_luatex:nn \use_ii:nn }
  }
```

`\xetex_or_luatex:nnn` Use #1 and ({#2} if XeTeX) or ({#3} if LuaTeX).

```
\xetex_if_engine:TF
  { \cs_new:Npn \xetex_or_luatex:nnn #1#2#3 {#1{#2}} }
  {
    \luatex_if_engine:T
     { \cs_new:Npn \xetex_or_luatex:nnn #1#2#3 {#1{#3}} }
  }
```

20.1 Bits and pieces

Conditionals

```
\newif\ifzf@firsttime
\newif\ifzf@nobf
\newif\ifzf@noit
\newif\ifzf@nosc
```

```
\newif\ifzf@tfm
\newif\ifzf@atsui
\newif\ifzf@icu
\newif\ifzf@mm
\newif\ifzf@graphite
```

For dealing with legacy maths

```
\newif\ifzf@math@euler
\newif\ifzf@math@lucida
\newif\ifzf@package@euler@loaded
```

For package options:

```
\newif\if@zf@configfile
\newif\if@zf@math
```

Counters

```
\newcount\c@zf@newff
\newcount\c@zf@index
\newcount\c@zf@script
\newcount\c@zf@language
\int_new:N \l_fontspec_strnum_int
```

Temporary definition until expl3 has been updated to include this:

```
\cs_set:Npn \use:x #1 { \edef\@tempa{#1}\@tempa }
```

```
\cs_set:Npn \use_v:nnnnn #1#2#3#4#5 {#5}
\cs_set:Npn \use_iv:nnnnn #1#2#3#4#5 {#4}
```

```
\cs_new:Npn \fontspec_setkeys:xx #1#2
 {
  \use:x { \exp_not:N \setkeys*[zf]{#1}{#2} }
 }
\cs_new:Npn \fontspec_setkeys:xxx #1#2#3
 {
  \use:x { \exp_not:N \setkeys*[zf@#1]{#2}{#3} }
 }
```

20.2 Error/warning messages

Shorthands for messages:

```
\cs_new:Npn \fontspec_error:n     { \msg_error:nn     {fontspec} }
\cs_new:Npn \fontspec_error:nx    { \msg_error:nnx    {fontspec} }
\cs_new:Npn \fontspec_warning:n   { \msg_warning:nn   {fontspec} }
\cs_new:Npn \fontspec_warning:nx  { \msg_warning:nnx  {fontspec} }
\cs_new:Npn \fontspec_warning:nxx { \msg_warning:nnxx {fontspec} }
\cs_new:Npn \fontspec_info:n      { \msg_info:nn      {fontspec} }
\cs_new:Npn \fontspec_info:nx     { \msg_info:nnx     {fontspec} }
\cs_new:Npn \fontspec_info:nxx    { \msg_info:nnxx    {fontspec} }
\cs_new:Npn \fontspec_trace:n     { \msg_trace:nn     {fontspec} }
```

Errors:

```
\msg_new:nnn {fontspec} {no-size-info}
{
```

```
  Size~ information~ must~ be~ supplied.\\
  For~ example,~ SizeFeatures={Size={8-12},...}.
}
\msg_new:nnnn {fontspec} {rename-feature-not-exist}
{
  The~ feature~ #1~ doesn't~ appear~ to~ be~ defined.
}
{
  It~ looks~ like~ you're~ trying~ to~ rename~ a~ feature~ that~ doesn't~ exist.
}
\msg_new:nnn {fontspec} {no-glyph}
{
  '\zf@fontname'~ doesn't~ appear~ to~ have~ the~ glyph~ corresponding~ to~ '#1'.
}
\msg_new:nnnn {fontspec} {unknown-options}
{
  The~ following~ font~ options~ are~ not~ recognised:\\
  \space\space\space\space #1
}
{
  There~ is~ probably~ a~ typo~ in~ the~ font~ feature~ selection.
}
\msg_new:nnnn {fontspec} {euler-too-late}
{
  The~ euler~ package~ must~ be~ loaded~ BEFORE~ fontspec.
}
{
  fontspec~ only~ overwrites~ euler's~ attempt~ to\\
  define~ the~ maths~ text~ fonts~ if~ fontspec~ is\\
  loaded~ after~ euler.~ Type~ return~ to~ proceed\\
  with~ incorrect~ \string\mathit,~ \string\mathbf,~ etc.
}
\msg_new:nnnn {fontspec} {no-xcolor}
{
  Cannot~ load~ named~ colours~ without~ the~ xcolor~ package.
}
{
  Sorry,~ I~ can't~ do~ anything~ to~ help.~ Instead~ of~ loading\\
  the~ color~ package,~ use~ xcolor~ instead.~ It's~ better.
}
\msg_new:nnnn {fontspec} {unknown-color-model}
{
  Error~ loading~ colour~ `#1';~ unknown~ colour~ model.
}
{
  Sorry,~ I~ can't~ do~ anything~ to~ help.~ Please~ report~ this~ error\\
  to~ my~ developer~ with~ a~ minimal~ example~ that~ causes~ the~ problem.
}
```

Warnings:

```
\msg_new:nnn {fontspec} {addfontfeatures-ignored}
{
```

```
  \string\addfontfeature (s)~ ignored;\\
  it~ cannot~ be~ used~ with~ a~ font~ that~ wasn't~ selected~ by~ fontspec.
}
\msg_new:nnn {fontspec} {feature-option-overwrite}
{
  Option~ '#2'~ of~ font~ feature~ '#1'~ overwritten.
}
\msg_new:nnn {fontspec} {script-not-exist}
{
  Font~ '\zf@fontname'~ does~ not~ contain~ script~ '#1'.
}
\msg_new:nnn {fontspec} {aat-feature-not-exist}
{
  '\XKV@tfam=\XKV@tkey'~ feature~ not~ supported\\
  for~ AAT~ font~ '\zf@fontname'.
}
\msg_new:nnn {fontspec} {aat-feature-not-exist-in-font}
{
  AAT~ feature~ '\XKV@tfam=\XKV@tkey'~ (#1)~ not~ available\\
  in~ font~ '\zf@fontname'.
}
\msg_new:nnn {fontspec} {icu-feature-not-exist}
{
  '\XKV@tfam=\XKV@tkey'~ feature~ not~ supported\\
  for~ ICU~ font~ '\zf@fontname'
}
\msg_new:nnn {fontspec} {icu-feature-not-exist-in-font}
{
  OpenType~ feature~ '\XKV@tfam=\XKV@tkey'~ (#1)~ not~ available\\
  for~ font~ '\zf@fontname', \\
  with~ script~ '\l_fontspec_script_name_tl',~ and~ language~ '\l_fontspec_lang_name_tl'.
}
\msg_new:nnn {fontspec} {no-opticals}
{
  '\zf@fontname'~ doesn't~ appear~ to~ have~ an~ Optical~ Size~ axis.
}
\msg_new:nnn {fontspec} {language-not-exist}
{
  Language~ '#1'~ not~ available\\
  for~ font~ '\zf@fontname'\\
  with~ script~ '\l_fontspec_script_name_tl'.
}
\msg_new:nnn {fontspec} {only-xetex-feature}
{
  Ignored~ XeTeX~ only~ feature:~ '#1'.
}
\msg_new:nnn {fontspec} {only-luatex-feature}
{
  Ignored~ LuaTeX~ only~ feature:~ '#1'.
}
\msg_new:nnn {fontspec} {no-mapping}
```

```
{
  Input~ mapping~ not~ (yet?)~ supported~ in~ LuaTeX.
}
\msg_new:nnn {fontspec} {no-mapping-ligtex}
{
  Input~ mapping~ not~ (yet?)~ supported~ in~ LuaTeX.\\
  Use~ "Ligatures=TeX"~ instead~ of~ "Mapping=tex-text".
}
\msg_new:nnn {fontspec} {cm-default-obsolete}
{
  The~ "cm-default"~ package~ option~ is~ obsolete.
}
```

Info messages:

```
\msg_new:nnn {fontspec} {defining-font}
{
  Defining~ font~ family~ for~ '#2'~ with~ options~ [\zf@default@options #1].
}
\msg_new:nnn {fontspec} {no-font-shape}
{
  Could~ not~ resolve~ font~ #1~ (it~ probably~ doesn't~ exist).
}
\msg_new:nnn {fontspec} {set-scale}
{
  \zf@fontname\space scale ~=~ \l_fontspec_scale_tl.
}
\msg_new:nnn {fontspec} {setup-math}
{
  Adjusting~ the~ maths~ setup~ (use~ [no-math]~ to~ avoid~ this).
}
\msg_new:nnn {fontspec} {no-scripts}
{
  Font~ \zf@fontname\space does~ not~ contain~ any~ OpenType~ 'Script'~ information.
}
```

20.3 Option processing

```
\DeclareOption{cm-default}{
  \fontspec_warning:n {cm-default-obsolete}
}
\DeclareOption{math}{\@zf@mathtrue}
\DeclareOption{no-math}{\@zf@mathfalse}
\DeclareOption{config}{\@zf@configfiletrue}
\DeclareOption{no-config}{\@zf@configfilefalse}
\DeclareOption{quiet}{
  \msg_redirect_module:nnn { fontspec } { warning } { info }
  \msg_redirect_module:nnn { fontspec } { info } { none }
}
\DeclareOption{silent}{
  \msg_redirect_module:nnn { fontspec } { warning } { none }
  \msg_redirect_module:nnn { fontspec } { info } { none }
}
```

```
\ExecuteOptions{config,math}
\ProcessOptions*
```

20.4 Packages

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

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

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

```
\RequirePackage{fontspec-patches}
```

20.5 Encodings

Frank Mittelbach has recommended using the 'EUx' family of font encodings to experiment with Unicode. Now that XETEX 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.

```
\xetex_or_luatex:nnn {\tl_set:Nn \zf@enc} {EU1} {EU2}
\tl_set:Nn \rmdefault {lmr}
\tl_set:Nn \sfdefault {lmss}
\tl_set:Nn \ttdefault {lmtt}
\RequirePackage[\zf@enc]{fontenc}
\tl_set_eq:NN \UTFencname \zf@enc % for xunicode
```

Dealing with a couple of the problems introduced by babel:

```
\tl_set_eq:NN \cyrillicencoding \zf@enc
\tl_set_eq:NN \latinencoding    \zf@enc
\g@addto@macro \document {
  \tl_set_eq:NN \cyrillicencoding \zf@enc
  \tl_set_eq:NN \latinencoding    \zf@enc
}
```

That latin encoding definition is repeated to suppress font warnings. Something to do with \select@language ending up in the .aux file which is read at the beginning of the document.

20.6 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 20.8 on page 58.

20.6.1 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.

```
\DeclareDocumentCommand \fontspec { O{} m } {
  \fontspec_set_family:Nnn \f@family {#1}{#2}
  \selectfont
  \ignorespaces
}
```

`\setmainfont` `\setsansfont` `\setmonofont`

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

```
\DeclareDocumentCommand \setmainfont { O{} m } {
  \fontspec_set_family:Nnn \rmdefault {#1}{#2}
  \normalfont
}
\DeclareDocumentCommand \setsansfont { O{} m } {
  \fontspec_set_family:Nnn \sfdefault {#1}{#2}
  \normalfont
}
\DeclareDocumentCommand \setmonofont { O{} m } {
  \fontspec_set_family:Nnn \ttdefault {#1}{#2}
  \normalfont
}
```

`\setromanfont`

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

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

`\setmathrm` `\setmathsf` `\setboldmathrm` `\setmathtt`

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

```
\DeclareDocumentCommand \setmathrm { O{} m } {
  \fontspec_set_family:Nnn \zf@rmmaths {#1}{#2}
}
\DeclareDocumentCommand \setboldmathrm { O{} m } {
  \fontspec_set_family:Nnn \zf@rmboldmaths {#1}{#2}
}
\DeclareDocumentCommand \setmathsf { O{} m } {
  \fontspec_set_family:Nnn \zf@sfmaths {#1}{#2}
}
\DeclareDocumentCommand \setmathtt { O{} m } {
  \fontspec_set_family:Nnn \zf@ttmaths {#1}{#2}
}
\@onlypreamble\setmathrm
\@onlypreamble\setboldmathrm
\@onlypreamble\setmathsf
\@onlypreamble\setmathtt
```

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

```
\def\zf@rmmaths{\rmdefault}
```

```
\def\zf@sfmaths{\sfdefault}
\def\zf@ttmaths{\ttdefault}
```

\newfontfamily
\newfontface

This macro takes the arguments of \fontspec with a prepended ⟨*instance cmd*⟩ (code for middle optional argument generated by Scott Pakin's newcommand.py). This command is used when a specific font instance needs to be referred to 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.

```
\DeclareDocumentCommand \newfontfamily { m O{} m } {
  \fontspec_select:nn{#2}{#3}
  \use:x {
    \exp_not:N \DeclareRobustCommand \exp_not:N #1 {
      \exp_not:N \fontfamily {\zf@family} \exp_not:N \selectfont
    }
  }
}
```

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

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

20.6.2 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.

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

```
\DeclareDocumentCommand \addfontfeatures {m} {
```

```
  \ifcsname zf@family@fontdef\f@family\endcsname
    \begingroup
      \let\zf@default@options\@empty
      \use:x {
        \exp_not:N\fontspec_select:nn
          {\csname zf@family@options\f@family\endcsname,#1}
          {\csname zf@family@fontname\f@family\endcsname}
      }
    \endgroup
    \fontfamily\zf@family\selectfont
  \else
    \fontspec_warning:n {addfontfeatures-ignored}
  \fi
  \ignorespaces
}
\let\addfontfeature\addfontfeatures
```

20.6.3 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.

```
\DeclareDocumentCommand \newfontfeature {mm} {
  \stepcounter{zf@newff}
  \cs_set:cpx{+zf-#1}{+zf-\the\c@zf@newff}
  \define@key[zf]{options}{#1}[]{
    \zf@update@family{\csname+zf-#1\endcsname}
    \zf@update@ff{#2}
  }
}
```

\newAATfeature

This command assigns a new AAT feature by its code (#2,#3) to a new name (#1). Better than \newfontfeature because it checks if the feature exists in the font it's being used for.

```
\DeclareDocumentCommand \newAATfeature {mmmm} {
  \unless\ifcsname zf@options@#1\endcsname
    \zf@define@font@feature{#1}
  \fi
  \key@ifundefined[zf]{#1}{#2}{}{
    \fontspec_warning:nxx {feature-option-overwrite}{#1}{#2}
  }
  \zf@define@feature@option{#1}{#2}{#3}{#4}{}
}
```

\newICUfeature
\newopentypefeature

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.

```
\DeclareDocumentCommand \newICUfeature {mmm} {
  \unless\ifcsname zf@options@#1\endcsname
    \zf@define@font@feature{#1}
  \fi
  \key@ifundefined[zf]{#1}{#2}{}{
    \fontspec_warning:nxx {feature-option-overwrite}{#1}{#2}
  }
  \zf@define@feature@option{#1}{#2}{}{}{#3}
}
\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.

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

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

```
\DeclareDocumentCommand \newfontscript {mm}
{
  \fontspec_new_script:nn {#1} {#2}
  \fontspec_new_script:nn {#2} {#2}
}
```

```
\cs_new:Npn \fontspec_new_script:nn #1#2
{
  \define@key[zf@feat]{Script}{#1}[]{
    \fontspec_check_script:nTF {#2} {
      \zf@update@family{+script=#1}
      \tl_set:Nn \l_fontspec_script_tl {#2}
      \c@zf@script=\l_fontspec_strnum_int\relax
    }{
      \fontspec_warning:nx {script-not-exist} {#1}
    }
  }
}
```

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

```
\DeclareDocumentCommand \newfontlanguage {mm}
{
  \fontspec_new_lang:nn {#1} {#2}
  \fontspec_new_lang:nn {#2} {#2}
}
```

```
\cs_new:Npn \fontspec_new_lang:nn #1#2
{
  \define@key[zf@feat]{Lang}{#1}[]{
    \fontspec_check_lang:nTF {#2} {
      \zf@update@family{+lang=#1}
      \tl_set:Nn \l_fontspec_lang_tl {#2}
      \c@zf@language=\l_fontspec_strnum_int\relax
    }{
      \fontspec_warning:nx {language-not-exist} {#1}
    }
  }
}
```

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

```
\DeclareDocumentCommand \DeclareFontsExtensions {m}
{
  \tl_set:Nx \l_fontspec_extensions_clist { \zap@space #1~\@empty }
}
\DeclareFontsExtensions{.otf,.ttf,.OTF,.TTF,.ttc,.TTC,.dfont}
```

20.7 Programmer's interface

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

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

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

`\fontspec_if_fontspec_font:TF` Returns whether the currently selected font has been loaded by fontspec.

```
\prg_new_conditional:Nnn \fontspec_if_fontspec_font: {TF,T,F} {
  \ifcsname zf@family@fontdef\f@family\endcsname
    \prg_return_true:
  \else
    \prg_return_false:
  \fi
}
```

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

```
\prg_new_conditional:Nnn \fontspec_if_aat_feature:nn {TF,T,F} {
  \ifcsname zf@family@fontdef\f@family\endcsname
    \font\zf@basefont="\use:c{zf@family@fontdef\f@family}"~at~\f@size pt
    \ifzf@atsui
      \fontspec_make_AAT_feature_string:nn{#1}{#2}
      \ifx\@tempa\@empty
        \prg_return_false:
      \else
```

```
        \prg_return_true:
      \fi
    \else
      \prg_return_false:
    \fi
  \else
    \prg_return_false:
  \fi
}
```

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

```
\prg_new_conditional:Nnn \fontspec_if_opentype: {TF,T,F} {
  \ifcsname zf@family@fontdef\f@family\endcsname
    \font\zf@basefont="\csname zf@family@fontdef\f@family\endcsname"~at~\f@size pt
    \fontspec_set_font_type:
    \ifzf@icu
      \prg_return_true:
    \else
      \prg_return_false:
    \fi
  \else
    \prg_return_false:
  \fi
}
```

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

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

`\fontspec_if_feature:nnnTF` Returns whether the currently selected font with raw OpenType script tag #1 and raw OpenType language tag #2 contains the raw OpenType feature tag #3. E.g.:

`\fontspec_if_feature:nTF {latn} {ROM} {pnum} {True} {False}` Returns false if the font is not loaded by fontspec or is not an OpenType font.

```
\prg_new_conditional:Nnn \fontspec_if_feature:nnn {TF,T,F} {
  \ifcsname zf@family@fontdef\f@family\endcsname
    \font\zf@basefont="\csname zf@family@fontdef\f@family\endcsname"~at~\f@size pt
    \fontspec_set_font_type:
    \ifzf@icu
      \fontspec_iv_str_to_num:n{#1} \c@zf@script   = \l_fontspec_strnum_int \relax
      \fontspec_iv_str_to_num:n{#2} \c@zf@language = \l_fontspec_strnum_int \relax
      \fontspec_check_ot_feat:nTF {#3} \prg_return_true: \prg_return_false:
    \else
      \prg_return_false:
    \fi
  \else
    \prg_return_false:
  \fi
}
```

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

```
\prg_new_conditional:Nnn \fontspec_if_script:n {TF,T,F} {
  \ifcsname zf@family@fontdef\f@family\endcsname
    \font\zf@basefont="\csname zf@family@fontdef\f@family\endcsname"~at~\f@size pt
    \fontspec_set_font_type:
    \ifzf@icu
      \fontspec_check_script:nTF {#1} \prg_return_true: \prg_return_false:
    \else
      \prg_return_false:
    \fi
  \else
    \prg_return_false:
  \fi
}
```

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

```
\prg_new_conditional:Nnn \fontspec_if_language:n {TF,T,F} {
  \ifcsname zf@family@fontdef\f@family\endcsname
    \font\zf@basefont="\csname zf@family@fontdef\f@family\endcsname"~at~\f@size pt
    \fontspec_set_font_type:
    \ifzf@icu
      \tl_set:Nv \l_fontspec_script_tl {g_fontspec_script_(\zf@family)_tl}
      \int_set:Nn \c@zf@script
        {\use:c {g_fontspec_script_num_(\zf@family)_tl}}
      \fontspec_check_lang:nTF {#1} \prg_return_true: \prg_return_false:
    \else
      \prg_return_false:
    \fi
  \else
```

```
    \prg_return_false:
  \fi
}
```

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

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

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

```
\prg_new_conditional:Nnn \fontspec_if_current_script:n {TF,T,F} {
  \ifcsname zf@family@fontdef\f@family\endcsname
    \font\zf@basefont="\csname zf@family@fontdef\f@family\endcsname"~at~\f@size pt
    \fontspec_set_font_type:
    \ifzf@icu
      \tl_if_eq:nvTF {#1} {g_fontspec_script_(\zf@family)_tl}
        {\prg_return_true:} {\prg_return_false:}
    \else
      \prg_return_false:
    \fi
  \else
    \prg_return_false:
  \fi
}
```

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

```
\prg_new_conditional:Nnn \fontspec_if_current_language:n {TF,T,F} {
  \ifcsname zf@family@fontdef\f@family\endcsname
    \font\zf@basefont="\csname zf@family@fontdef\f@family\endcsname"~at~\f@size pt
    \fontspec_set_font_type:
    \ifzf@icu
      \tl_if_eq:nvTF {#1} {g_fontspec_lang_(\zf@family)_tl}
        {\prg_return_true:} {\prg_return_false:}
    \else
      \prg_return_false:
```

```
    \fi
  \else
    \prg_return_false:
  \fi
}
```

Need this:

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

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

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

We want to store the actual name of the font family within the ⟨*family*⟩ variable because the actual LaTeX 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.

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

20.8 Internal macros

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

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

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

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

```
  \edef\zf@fontname{#2}
  \let\zf@family@long\zf@fontname
  \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 occuring per-shape this no longer needs to happen; this is indicated by the 'firsttime' conditional.

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

```
\fontspec_save_family:nT {#2} {
  \fontspec_info:nxx {defining-font} {#1} {#2}
  \fontspec_save_fontinfo:nn {#1} {#2}
  \DeclareFontFamily{\zf@enc}{\zf@family}{}
  \fontspec_set_upright:
  \fontspec_set_bold:
  \fontspec_set_italic:
  \fontspec_set_slanted:
  \fontspec_set_bold_italic:
  \fontspec_set_bold_slanted:
}
\endgroup
}
```

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

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

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

```
\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.

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

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

Finally save the 'confirmed' font definition.

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

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

```
\prg_new_conditional:Nnn \fontspec_if_detect_external:n {T}
{
  \clist_map_inline:Nn \l_fontspec_extensions_clist
   {
    \bool_set_false:N \l_tmpa_bool
    \tl_if_in:nnT {#1 = end_of_string} {##1 = end_of_string}
      { \bool_set_true:N \l_tmpa_bool \clist_map_break: }
   }
  \bool_if:NTF \l_tmpa_bool \prg_return_true: \prg_return_false:
}
```

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

```
\cs_set:Npn \fontspec_fullname:n #1 {
  \fontspec_namewrap:n { #1 \l_fontspec_extension_tl }
  \l_fontspec_renderer_tl
  \l_fontspec_optical_size_tl
}
```

`\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.

```
\prg_new_conditional:Nnn \fontspec_save_family:n {T} {
  \unless\ifcsname zf@UID@\zf@family@long\endcsname
    \ifcsname c@zf@famc@#1\endcsname
      \expandafter\global\expandafter\advance
        \csname c@zf@famc@#1\endcsname\@ne
    \else
      \expandafter\global\expandafter\newcount
        \csname c@zf@famc@#1\endcsname
    \fi
    \edef\@tempa{#1~}
    \cs_gset:cpx{zf@UID@\zf@family@long}{
      \expandafter\zap@space\@tempa\@empty
      (\expandafter\the\csname c@zf@famc@#1\endcsname)
    }
  \fi
  \xdef\zf@family{\@nameuse{zf@UID@\zf@family@long}}
  \cs_if_exist:cTF {zf@family@fontname\zf@family}
    \prg_return_false: \prg_return_true:
}
```

`\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.

```
\cs_new:Npn \fontspec_set_scriptlang: {
  \ifzf@icu
    \tl_if_empty:NTF \l_fontspec_script_name_tl {
      \fontspec_check_script:nTF {latn}
      {
        \tl_set:Nn \l_fontspec_script_name_tl {Latin}
        \tl_set:Nn \l_fontspec_script_tl      {latn}
        \tl_if_empty:NT \l_fontspec_lang_tl {
          \tl_set:Nn \l_fontspec_lang_name_tl {Default}
          \tl_set:Nn \l_fontspec_lang_tl      {DFLT}
        }
        \fontspec_setkeys:xxx {feat} {Script} {\l_fontspec_script_name_tl}
        \fontspec_setkeys:xxx {feat} {Lang}   {\l_fontspec_lang_name_tl}
      }
      {
        \fontspec_info:n {no-scripts}
      }
    }
    {
      \tl_if_empty:NT \l_fontspec_lang_tl {
        \tl_set:Nn \l_fontspec_lang_name_tl {Default}
        \tl_set:Nn \l_fontspec_lang_tl      {DFLT}
      }
      \fontspec_setkeys:xxx {feat} {Script} {\l_fontspec_script_name_tl}
      \fontspec_setkeys:xxx {feat} {Lang}   {\l_fontspec_lang_name_tl}
    }
  \fi
}
```

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

```
\cs_new:Npn \fontspec_save_fontinfo:nn #1#2 {
    \tl_gset:cx {zf@family@fontname\zf@family} {#2}
    \tl_gset:cx {zf@family@options\zf@family} {\zf@default@options #1}
    \tl_gset:cx {zf@family@fontdef\zf@family} {
      \fontspec_fullname:n {\zf@fontname} : \l_fontspec_pre_feat_tl \l_fontspec_rawfeatures_sclist
    }
    \tl_gset:cx {g_fontspec_script_num_(\zf@family)_tl}
      { \int_use:N \c@zf@script   }
    \tl_gset:cx {g_fontspec_lang_num_(\zf@family)_tl}
      { \int_use:N \c@zf@language }
    \tl_gset_eq:cN {g_fontspec_script_(\zf@family)_tl} \l_fontspec_script_tl
    \tl_gset_eq:cN {g_fontspec_lang_(\zf@family)_tl}   \l_fontspec_lang_tl
}
```

`\fontspec_set_upright:` Sets the upright shape.

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

```
}
```

`\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.

```
\cs_new:Npn \fontspec_set_bold: {
  \unless\ifzf@nobf
    \ifx\zf@bf\@empty
      \zf@make@font@shapes[\zf@fontname]{/B}
        {\bfdefault}{\updefault}{\zf@font@feat\zf@bf@feat}
    \else
      \zf@make@font@shapes{\zf@bf}
        {\bfdefault}{\updefault}{\zf@font@feat\zf@bf@feat}
    \fi
  \fi
}
```

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

```
\cs_new:Npn \fontspec_set_italic: {
  \unless\ifzf@noit
    \ifx\zf@it\@empty
      \zf@make@font@shapes[\zf@fontname]{/I}
        {\mddefault}{\itdefault}{\zf@font@feat\zf@it@feat}
    \else
      \zf@make@font@shapes{\zf@it}
        {\mddefault}{\itdefault}{\zf@font@feat\zf@it@feat}
    \fi
  \fi
}
```

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

```
\cs_new:Npn \fontspec_set_slanted: {
  \ifx\zf@sl\@empty\else
    \zf@make@font@shapes{\zf@sl}
      {\mddefault}{\sldefault}{\zf@font@feat\zf@sl@feat}
  \fi
}
```

`\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 occurance, presumably), the new bold font is used to define the new bold italic font.

```
\cs_new:Npn \fontspec_set_bold_italic: {
  \@tempswatrue
  \ifzf@nobf\@tempswafalse\fi
  \ifzf@noit\@tempswafalse\fi
  \bool_if:NT \l_fontspec_external_bool \@tempswafalse
  \if@tempswa
```

```
    \ifx\zf@bfit\@empty
      \ifx\zf@bf\@empty
        \ifx\zf@it\@empty
          \zf@make@font@shapes[\zf@fontname]{/BI}
            {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}
        \else
          \zf@make@font@shapes[\zf@it]{/B}
            {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}
        \fi
      \else
        \zf@make@font@shapes[\zf@bf]{/I}
          {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}
      \fi
    \else
      \zf@make@font@shapes{\zf@bfit}
        {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}
    \fi
  \fi
}
```

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

```
\cs_new:Npn \fontspec_set_bold_slanted: {
  \ifx\zf@bfsl\@empty
    \ifx\zf@sl\@empty\else
      \zf@make@font@shapes[\zf@sl]{/B}
        {\bfdefault}{\sldefault}{\zf@font@feat\zf@bfsl@feat}
    \fi
  \else
    \zf@make@font@shapes{\zf@bfsl}
      {\bfdefault}{\sldefault}{\zf@font@feat\zf@bfsl@feat}
  \fi
}
```

20.8.1 Fonts

`\fontspec_set_font_type:` Now check if the font is to be rendered with ATSUI or ICU. This will either be automatic (based on the font type), or specified by the user via a font feature.

This macro sets `\zf@atsui` or `\zf@icu` or `\zf@mm` booleans accordingly depending if the font in `\zf@basefont` is an AAT font or an OpenType font or a font with feature axes (either AAT or Multiple Master), respectively.

```
\xetex_or_luatex:nnn { \cs_new:Npn \fontspec_set_font_type: }
  {
    \zf@tfmfalse \zf@atsuifalse \zf@icufalse \zf@mmfalse \zf@graphitefalse
    \ifcase\XeTeXfonttype\zf@basefont
      \zf@tfmtrue
    \or
      \zf@atsuitrue
      \ifnum\XeTeXcountvariations\zf@basefont  \c_zero
        \zf@mmtrue
      \fi
    \or
```

```
      \zf@icutrue
    \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.

```
    \tl_if_empty:NT \l_fontspec_renderer_tl {
      \ifzf@atsui
        \tl_set:Nn \l_fontspec_renderer_tl {/AAT}
      \else\ifzf@icu
        \tl_set:Nn \l_fontspec_renderer_tl {/ICU}
      \fi\fi
    }
  }
  {
    \zf@icutrue
  }
```

`\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 XeTeX'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.

```
\newcommand*\zf@make@font@shapes[5][]{
  \begingroup
    \edef\@tempa{#1}
    \unless\ifx\@tempa\@empty
      \font\@tempfonta="\fontspec_fullname:n {#1}"~at~\f@size pt
      \edef\@tempa{\fontname\@tempfonta}
    \fi
    \font\@tempfontb="\fontspec_fullname:n {#1#2}"~at~\f@size pt
    \edef\@tempb{\fontname\@tempfontb}
    \ifx\@tempa\@tempb
      \fontspec_info:nx {no-font-shape} {#1#2}
    \else
      \edef\zf@fontname{#1#2}
      \let\zf@basefont\@tempfontb
      \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 20.10 on page 93 for the code that enables this usage.

```
      \ifx\zf@sc\@empty
        \unless\ifzf@nosc
          \zf@make@smallcaps
          \unless\ifx\zf@smallcaps\@empty
            \fontspec_declare_shape:nnnn {\zf@smallcaps}{#3}
              {\ifx#4\itdefault\sidefault\else\scdefault\fi}{#5\zf@sc@feat}
          \fi
        \fi
      \else
        \edef\zf@fontname{\zf@sc}
        \fontspec_declare_shape:nnnn {}{#3}
          {\ifx#4\itdefault\sidefault\else\scdefault\fi}{#5\zf@sc@feat}
      \fi
    \fi
  \endgroup
}
```

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`.

```
\cs_new:Npn \fontspec_declare_shape:nnnn #1#2#3#4 {
  \clist_if_empty:NTF \l_fontspec_sizefeat_clist
  {
    \fontspec_get_features:n{#4}
    \tl_set:Nx \l_fontspec_nfss_tl {
      - \l_fontspec_scale_tl "
      \fontspec_fullname:n {\zf@fontname} :
        \l_fontspec_pre_feat_tl \l_fontspec_rawfeatures_sclist #1 "
    }
  }
```

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}}`.

```
  {
    \tl_clear:N \l_fontspec_nfss_tl
    \clist_map_inline:Nn \l_fontspec_sizefeat_clist {
      \tl_clear:N \l_fontspec_size_tl
      \tl_set_eq:NN \l_fontspec_sizedfont_tl \zf@fontname
      \fontspec_setkeys:xx {sizing} { \expandafter \@firstofone ##1 }
      \tl_if_empty:NT \l_fontspec_size_tl { \fontspec_error:n {no-size-info} }
      \fontspec_get_features:n{#4,\XKV@rm}
      \tl_put_right:Nx \l_fontspec_nfss_tl {
        \l_fontspec_size_tl \l_fontspec_scale_tl
```

```
        " \fontspec_fullname:n { \l_fontspec_sizedfont_tl }
          : \l_fontspec_pre_feat_tl \l_fontspec_rawfeatures_sclist #1 "
      }
    }
  }
```

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.

```
  \PackageInfo{fontspec}{
    Defining~ shape~ #2/#3~ with~ raw~ font~ features:
    \MessageBreak \l_fontspec_rawfeatures_sclist
  \@gobble}
  \use:x{
    \exp_not:N\DeclareFontShape{\zf@enc}{\zf@family}{#2}{#3}
      {\l_fontspec_nfss_tl}{\zf@adjust}
  }
```

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?

```
  \tl_if_eq:xxT {#3} {\itdefault}
  {
    \use:x {
      \exp_not:N \DeclareFontShape {\zf@enc}{\zf@family}{#2}{\sldefault}
        {-ssub*\zf@family/#2/\itdefault}{\zf@adjust}
    }
  }
}
```

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

```
\xetex_or_luatex:nnn { \tl_set:Nn \l_fontspec_pre_feat_tl }
  {
    \ifzf@icu
      \tl_if_empty:NF \l_fontspec_script_tl
      {
        script   = \l_fontspec_script_tl ;
        language = \l_fontspec_lang_tl   ;
      }
    \fi
  }
  {
    mode     = \l_fontspec_mode_tl   ;
    \tl_if_empty:NF \l_fontspec_script_tl
    {
      script   = \l_fontspec_script_tl ;
      language = \l_fontspec_lang_tl   ;
    }
  }
```

`\zf@update@family` This macro is used to build up a complex family name based on its features.

`\zf@firsttime` is set true in `\fontspec_select:nn` only the first time `\f@get@feature@requests` is called, so that the family name is only created once.

```
\newcommand*{\zf@update@family}[1]{
  \ifzf@firsttime
    \xdef\zf@family@long{\zf@family@long#1}
  \fi
}
```

20.8.2 Features

`\fontspec_get_features:n` This macro is a wrapper for `\setkeys` which expands and adds a default specification to the original passed options. It begins by initialising the commands used to hold font-feature specific strings.

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

Finish the colour specification:

```
  \cs_if_exist:NT \l_fontspec_hexcol_tl {
    \zf@update@ff{color=\l_fontspec_hexcol_tl\l_fontspec_opacity_tl}
  }
}
```

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

```
\tl_clear:N \zf@bf
\tl_clear:N \zf@it
\tl_clear:N \zf@fake@slant
\tl_clear:N \zf@fake@embolden
\tl_clear:N \zf@bfit
\tl_clear:N \zf@sl
\tl_clear:N \zf@bfsl
\tl_clear:N \zf@sc
\tl_clear:N \zf@up@feat
\tl_clear:N \zf@bf@feat
\tl_clear:N \zf@it@feat
\tl_clear:N \zf@bfit@feat
\tl_clear:N \zf@sl@feat
\tl_clear:N \zf@bfsl@feat
\tl_clear:N \zf@sc@feat
\tl_clear:N \l_fontspec_script_name_tl
\tl_clear:N \l_fontspec_script_tl
\tl_clear:N \l_fontspec_lang_name_tl
\tl_clear:N \l_fontspec_lang_tl
\clist_clear:N \l_fontspec_sizefeat_clist
\tl_set:Nn \l_fontspec_opacity_tl {FF}
```

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

```
\newcommand*\fontspec_init:{
  \zf@icufalse
  \zf@firsttimetrue
  \xetex_or_luatex:nnn { \cs_set:Npn \fontspec_namewrap:n ##1 }
    { ##1 }
    { name:##1 }
  \tl_clear:N \l_fontspec_optical_size_tl
  \tl_clear:N \l_fontspec_renderer_tl
  \luatex_if_engine:T {
    \tl_set:Nn \l_fontspec_mode_tl {node}
    \luatexprehyphenchar   =`\-  % fixme
    \luatexposthyphenchar  = 0   % fixme
    \luatexpreexhyphenchar = 0   % fixme
    \luatexpostexhyphenchar= 0   % fixme
  }
}
```

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

```
\newcommand*\zf@make@smallcaps{
  \let\zf@smallcaps\@empty
  \xetex_or_luatex:nn
  {
    \ifzf@atsui
      \fontspec_make_AAT_feature_string:nn{3}{3}
      \unless\ifx\@tempa\@empty
        \edef\zf@smallcaps{\@tempa;}
      \fi
    \fi
    \ifzf@icu
      \fontspec_check_ot_feat:nT {+smcp} {\edef\zf@smallcaps{+smcp;}}
    \fi
  }
  {
    \fontspec_check_ot_feat:nT {+smcp} {\edef\zf@smallcaps{+smcp;}}
  }
}
```

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

```
\cs_new:Npn \sclist_put_right:Nn #1#2 {
  \tl_if_empty:NTF #1 {
    \tl_set:Nn #1 {#2}
  }{
    \tl_put_right:Nn #1 {;#2}
  }
}
```

`\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.

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

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

```
\cs_new:Npn \fontspec_make_feature:nnn #1#2#3 {
  \xetex_or_luatex:nn
  {
    \ifzf@atsui
      \fontspec_make_AAT_feature:nn {#1}{#2}
    \fi
    \ifzf@icu
      \fontspec_make_ICU_feature:n {#3}
    \fi
  }
  {
    \fontspec_make_ICU_feature:n {#3}
  }
}
\cs_generate_variant:Nn \fontspec_make_feature:nnn {nnx}
\cs_new:Npn \fontspec_make_AAT_feature:nn #1#2 {
  \tl_if_empty:nTF {#1}
  {
    \fontspec_warning:n {aat-feature-not-exist}
  }
  {
    \fontspec_make_AAT_feature_string:nn {#1}{#2}
    \ifx\@tempa\@empty
      \fontspec_warning:nx {aat-feature-not-exist-in-font} {#1,#2}
    \else
      \zf@update@family{+#1,#2}
      \zf@update@ff\@tempa
    \fi
  }
}
\cs_new:Npn \fontspec_make_ICU_feature:n #1 {
  \tl_if_empty:nTF {#1}
  {
    \fontspec_warning:n {icu-feature-not-exist}
  }
  {
    \fontspec_check_ot_feat:nTF {#1} {
      \zf@update@family{#1}
      \zf@update@ff{#1}
    }{
      \fontspec_warning:nx {icu-feature-not-exist-in-font} {#1}
    }
```

```
  }
}
```

`\zf@define@font@feature`
`\zf@define@feature@option`

These macros are used in order to simplify font feature definition later on.

```
\newcommand*\zf@define@font@feature[1]{
  \define@key[zf]{options}{#1}{{\setkeys[zf@feat]{#1}{##1}}}
}
\newcommand*\zf@define@feature@option[5]{
  \define@key[zf@feat]{#1}{#2}[]{\fontspec_make_feature:nnn{#3}{#4}{#5}}
}
```

`\keyval@alias@key`

This macro maps one xkeyval key to another.

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

`\multi@alias@key`

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

```
\newcommand*\multi@alias@key[2]{
  \key@ifundefined[zf]{options}{#1}
    {
      \key@ifundefined[zf]{preparse}{#1}
      {
        \key@ifundefined[zf]{preparse-external}{#1}
          { \fontspec_warning:nx {rename-feature-not-exist} {#1} }
          { \keyval@alias@key[zf]{preparse-external}{#1}{#2} }
      }
      { \keyval@alias@key[zf]{preparse}{#1}{#2} }
    }
    { \keyval@alias@key[zf]{options}{#1}{#2} }
}
```

`spec_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 68).

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

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

```
    \ifnum\XeTeXisexclusivefeature\zf@basefont #10
      \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:

```
    \else
      \unless\ifodd #2
        \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.

```
    \else
      \edef\@tempb{
        \XeTeXselectorname\zf@basefont #1\space \numexpr#2-1\relax
      }
      \unless\ifx\@tempb\@empty
        \edef\@tempb{!\@tempb}
      \fi
    \fi
  \fi
```

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

```
  \unless\ifx\@tempb\@empty
    \edef\@tempa{\@tempa=\@tempb}
  \else
    \let\@tempa\@empty
  \fi
\fi
}
```

`\fontspec_iv_str_to_num:n`
`\fontspec_v_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', 'abc ', 'ab', 'ab ', *etc*. (It is assumed the first two chars are *always* not spaces.) So this macro reads in the string, delimited by a space; this input is padded with `\@empty`s and anything beyond four chars is snipped. The `\@empty`s then are used to reconstruct the spaces in the string to number calculation.

The variant `\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.

```
\cs_set:Npn \fontspec_iv_str_to_num:n #1 {
  \fontspec_iv_str_to_num:w #1 \@empty \@empty \q_nil
}
\cs_set:Npn \fontspec_iv_str_to_num:w #1#2#3#4#5 \q_nil {
  \int_set:Nn \l_fontspec_strnum_int {
      `#1 * "1000000
    + `#2 * "10000
    + \ifx \@empty #3 32 \else `#3 \fi * "100
    + \ifx \@empty #4 32 \else `#4 \fi
  }
}
```

```
\cs_set:Npn \fontspec_v_str_to_num:n #1 {
  \bool_if:nTF
    {
      \tl_if_head_eq_charcode_p:nN {#1} {+} ||
      \tl_if_head_eq_charcode_p:nN {#1} {-}
    }
    {
      \exp_after:wN \fontspec_iv_str_to_num:n
      \exp_after:wN { \use_none:n #1 }
    }
    { \fontspec_iv_str_to_num:n {#1} }
}
```

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

```
\xetex_or_luatex:nnn {\prg_new_conditional:Nnn \fontspec_check_script:n {TF}}
  {
    \fontspec_iv_str_to_num:n{#1}
    \@tempcntb\XeTeXOTcountscripts\zf@basefont
    \c@zf@index\z@ \@tempswafalse
    \loop\ifnum\c@zf@index\@tempcntb
      \ifnum\XeTeXOTscripttag\zf@basefont\c@zf@index=\l_fontspec_strnum_int
        \@tempswatrue
        \c@zf@index\@tempcntb
      \else
        \advance\c@zf@index\@ne
      \fi
    \repeat
    \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
  }
  {
    \directlua{fontspec.check_ot_script("zf@basefont", "#1")}
    \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
  }
```

`\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 `\@tempswatrue`. `\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'.

```
\xetex_or_luatex:nnn {\prg_new_conditional:Nnn \fontspec_check_lang:n {TF}}
  {
    \fontspec_iv_str_to_num:n{#1}
    \@tempcntb\XeTeXOTcountlanguages\zf@basefont\c@zf@script
    \c@zf@index\z@
    \@tempswafalse
    \loop\ifnum\c@zf@index\@tempcntb
      \ifnum\XeTeXOTlanguagetag\zf@basefont\c@zf@script\c@zf@index=\l_fontspec_strnum_int
        \@tempswatrue
        \c@zf@index\@tempcntb
```

```
      \else
        \advance\c@zf@index\@ne
      \fi
    \repeat
    \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
  }
  {
    \directlua{
      fontspec.check_ot_lang( "zf@basefont", "#1", "\l_fontspec_script_tl" )
    }
    \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
  }
```

`\fontspec_check_ot_feat:nTF` `\fontspec_check_ot_feat:nT`

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'.

```
\xetex_or_luatex:nnn
  { \prg_new_conditional:Nnn \fontspec_check_ot_feat:n {TF,T} }
  {
    \@tempcntb\XeTeXOTcountfeatures\zf@basefont\c@zf@script\c@zf@language
    \fontspec_v_str_to_num:n {#1}
    \c@zf@index\z@
    \@tempswafalse
    \loop\ifnum\c@zf@index\@tempcntb
      \ifnum\XeTeXOTfeaturetag\zf@basefont\c@zf@script\c@zf@language
            \c@zf@index=\l_fontspec_strnum_int
        \@tempswatrue
        \c@zf@index\@tempcntb
      \else
        \advance\c@zf@index\@ne
      \fi
    \repeat
    \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
  }
  {
    \directlua{
      fontspec.check_ot_feat(
                                "zf@basefont", "#1",
                                "\l_fontspec_lang_tl", "\l_fontspec_script_tl"
                              )
    }
    \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
  }
```

20.9 keyval definitions

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

20.9.1 Pre-parsing naming information

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

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

```
\bool_new:N \l_fontspec_external_bool
\define@key[zf]{preparse-external}{ExternalLocation}[]{
  \zf@nobftrue
  \zf@noittrue
  \bool_set_true:N \l_fontspec_external_bool
  \xetex_or_luatex:nnn { \cs_gset:Npn \fontspec_namewrap:n ##1 }
    {     [ #1 ##1 ] }
    { file: #1 ##1   }
  \xetex_if_engine:T { \setkeys[zf]{preparse}{Renderer=ICU} }
}
\aliasfontfeature{ExternalLocation}{Path}
```

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

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

20.9.2 Pre-parsed features

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

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

```
\define@choicekey[zf]{preparse}{Renderer}[\l_tmpa_tl\l_tmpa_num]
    {AAT,ICU,Graphite,Full,Basic}{
  \zf@update@family{+rend:#1}
  \intexpr_compare:nTF {\l_tmpa_num  3} {
    \xetex_or_luatex:nn
    {
      \tl_set:Nv \l_fontspec_renderer_tl {g_fontspec_renderer_tag_\l_tmpa_tl}
    }
    {
```

```
      \fontspec_warning:nx {only-xetex-feature} {Renderer=AAT/ICU/Graphite}
    }
  }{
    \xetex_or_luatex:nn
    { \fontspec_warning:nx {only-luatex-feature} {Renderer=Full/Basic} }
    { \tl_set:Nv \l_fontspec_mode_tl {g_fontspec_mode_tag_\l_tmpa_tl}  }
  }
}
\tl_set:cn {g_fontspec_renderer_tag_AAT} {/AAT}
\tl_set:cn {g_fontspec_renderer_tag_ICU} {/ICU}
\tl_set:cn {g_fontspec_renderer_tag_Graphite} {/GR}
\tl_set:cn {g_fontspec_mode_tag_Full}  {node}
\tl_set:cn {g_fontspec_mode_tag_Basic} {base}
```

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

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

Exactly the same:

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

20.9.3 Bold/italic choosing options

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

Fonts Upright:

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

Bold:

```
\define@key[zf]{preparse-external}{BoldFont}{
  \edef\@tempa{#1}
  \ifx\@tempa\@empty
    \zf@nobftrue
    \zf@update@family{nobf}
  \else
    \zf@nobffalse
    \fontspec_complete_fontname:Nn \zf@bf {#1}
    \zf@update@family{bf:#1}
  \fi
```

```
}
```

Same for italic:

```
\define@key[zf]{preparse-external}{ItalicFont}{
  \edef\@tempa{#1}
  \ifx\@tempa\@empty
    \zf@noittrue
    \zf@update@family{noit}
  \else
    \zf@noitfalse
    \fontspec_complete_fontname:Nn \zf@it {#1}
    \zf@update@family{it:#1}
  \fi
}
```

Simpler for bold+italic & slanted:

```
\define@key[zf]{preparse-external}{BoldItalicFont}{
  \fontspec_complete_fontname:Nn \zf@bfit {#1}
  \zf@update@family{bfit:#1}
}
\define@key[zf]{preparse-external}{SlantedFont}{
  \fontspec_complete_fontname:Nn \zf@sl {#1}
  \zf@update@family{sl:#1}
}
\define@key[zf]{preparse-external}{BoldSlantedFont}{
  \fontspec_complete_fontname:Nn \zf@bfsl {#1}
  \zf@update@family{bfsl:#1}
}
```

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

```
\define@key[zf]{options}{SmallCapsFont}{
  \edef\@tempa{#1}
  \ifx\@tempa\@empty
    \zf@nosctrue
    \zf@update@family{nosc}
  \else
    \zf@noscfalse
    \fontspec_complete_fontname:Nn \zf@sc {#1}
    \zf@update@family{sc:\zap@space #1~\@empty}
  \fi
}
```

`\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`").

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

Features

```
\define@key[zf]{preparse}{UprightFeatures}{
  \def\zf@up@feat{, #1}
  \zf@update@family{rmfeat:#1}
}
\define@key[zf]{preparse}{BoldFeatures}{
  \def\zf@bf@feat{, #1}
  \zf@update@family{bffeat:#1}
}
\define@key[zf]{preparse}{ItalicFeatures}{
  \def\zf@it@feat{, #1}
  \zf@update@family{itfeat:#1}
}
\define@key[zf]{preparse}{BoldItalicFeatures}{
  \def\zf@bfit@feat{, #1}
  \zf@update@family{bfitfeat:#1}
}
\define@key[zf]{preparse}{SlantedFeatures}{
  \def\zf@sl@feat{, #1}
  \zf@update@family{slfeat:#1}
}
\define@key[zf]{preparse}{BoldSlantedFeatures}{
  \def\zf@bfsl@feat{, #1}
  \zf@update@family{bfslfeat:#1}
}
```

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

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

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

```
\define@key[zf]{preparse}{SizeFeatures}{
  \tl_set:Nn \l_fontspec_sizefeat_clist {#1}
  \zf@update@family{sizefeat:\zap@space #1~\@empty}
}
```

```
\define@key[zf]{sizing}{Size}{ \tl_set:Nn \l_fontspec_size_tl {#1} }
\define@key[zf]{sizing}{Font}{
  \fontspec_complete_fontname:Nn \l_fontspec_sizedfont_tl {#1}
}
```

20.9.4 Font-independent features

These features can be applied to any font.

Scale If the input isn't one of the pre-defined string options, then it's gotta be numerical. `\fontspec_calc_scale:n` does all the work in the auto-scaling cases.

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

```
    {MatchLowercase} { \fontspec_calc_scale:n {5} }
    {MatchUppercase} { \fontspec_calc_scale:n {8} }
  }
  { \tl_set:Nx \l_fontspec_scale_tl {#1} }
  \zf@update@family{+scale:\l_fontspec_scale_tl}
  \tl_set:Nx \l_fontspec_scale_tl { s*[\l_fontspec_scale_tl] }
}
```

`\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 XeTeX).

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.

```
\cs_new:Npn \fontspec_calc_scale:n #1 {
  \group_begin:
    \rmfamily
    \fontspec_set_font_dimen:NnN \@tempdima {#1} \font
    \fontspec_set_font_dimen:NnN \@tempdimb {#1} \zf@basefont
    \dim_set:Nn \@tempdimc { 1pt*\@tempdima/\@tempdimb }
    \tl_gset:Nx \l_fontspec_scale_tl {\strip@pt\@tempdimc}
    \fontspec_info:n {set-scale}
  \group_end:
}
```

`\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'.

```
\cs_new:Npn \fontspec_set_font_dimen:NnN #1#2#3
{
  \dim_set:Nn #1 { \fontdimen #2 #3 }
  \dim_compare:nNnT #1 = {0pt} {
    \settoheight #1 {
      \tl_if_eq:nnTF {#3} {\font} \rmfamily #3
      \prg_case_int:nnn #2 {
        {5} {x} % x-height
        {8} {X} % cap-height
      } {?} % "else" clause; never reached.
    }
  }
}
```

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

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

```
    \zf@wordspace@parse#1,\zf@@ii,\zf@@iii,\zf@@
  \fi
}
```

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

```
\def\zf@wordspace@parse#1,#2,#3,#4\zf@@{
  \def\@tempa{#4}
  \ifx\@tempa\@empty
    \setlength\@tempdima{#1\fontdimen2\zf@basefont}
    \@tempdimb\@tempdima
    \@tempdimc\@tempdima
  \else
    \setlength\@tempdima{#1\fontdimen2\zf@basefont}
    \setlength\@tempdimb{#2\fontdimen3\zf@basefont}
    \setlength\@tempdimc{#3\fontdimen4\zf@basefont}
  \fi
  \edef\zf@adjust{
    \zf@adjust
    \fontdimen2\font\the\@tempdima
    \fontdimen3\font\the\@tempdimb
    \fontdimen4\font\the\@tempdimc
  }
}
```

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

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

Letterspacing

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

Hyphenation character This feature takes one of three arguments: 'None', ⟨*glyph*⟩, or ⟨*slot*⟩. If the input isn't the first, and it's one character, then it's the second; otherwise, it's the third.

```
\define@key[zf]{options}{HyphenChar}{
  \zf@update@family{+hyphenchar:#1}
  \edef\@tempa{#1}
  \edef\@tempb{None}
  \ifx\@tempa\@tempb
    \xetex_or_luatex:nnn { \g@addto@macro\zf@adjust }
      { \hyphenchar\font-1\relax }
      { \luatexprehyphenchar=-1\relax }
```

```
  \else
    \zf@check@one@char#1\zf@@
    \ifx\@tempb\@empty
      \xetex_or_luatex:nn {
        {\zf@basefont\expandafter\ifnum\expandafter\XeTeXcharglyph
              \expandafter`#1  \z@
           \g@addto@macro\zf@adjust{%
             {\expandafter\hyphenchar\expandafter
              \font\expandafter`#1}}%
        \else
          \fontspec_error:nx {no-glyph}{#1}
        \fi}%
      }{
        \ifnum\directlua{
            fontspec.sprint(fontspec.charglyph('#1', 'zf@basefont'))
        }  \z@
          \g@addto@macro\zf@adjust{\luatexprehyphenchar=`#1\relax}
        \else
          \fontspec_error:nx {no-glyph}{#1}
        \fi
      }
    \else
      \xetex_or_luatex:nn {
        {\zf@basefont\ifnum\XeTeXcharglyph#1  \z@
          \g@addto@macro\zf@adjust{\hyphenchar\font#1\relax}%
        \else
          \fontspec_error:nx {no-glyph}{#1}
        \fi}%
      }{
        \ifnum\directlua{
            fontspec.sprint(fontspec.charglyph('#1', 'zf@basefont'))
        }  \z@
          \g@addto@macro\zf@adjust{\luatexprehyphenchar=#1\relax}
        \else
          \fontspec_error:nx {no-glyph}{#1}
        \fi
      }
    \fi
  \fi
}
\def\zf@check@one@char#1#2\zf@@{\def\@tempb{#2}}
```

Color

```
\define@key[zf]{options}{Color}{
  \zf@update@family{+col:#1}
  \cs_if_exist:cTF {\token_to_str:N\color@#1}
  {
    \convertcolorspec{named}{#1}{HTML}\l_fontspec_hexcol_tl
  }
  {
    \tl_set:Nn \l_fontspec_hexcol_tl {#1}
```

```
  }
}
\keyval@alias@key[zf]{options}{Color}{Colour}
\newcounter{fontspec_tmp_int}
\define@key[zf]{options}{Opacity}{
  \zf@update@family{+opac:#1}
  \setcounter {fontspec_tmp_int} { 255*\real{#1} }
  \tl_set:Nx \l_fontspec_opacity_tl {
    \nhex2 { \value{fontspec_tmp_int} }
  }
}
```

Mapping

```
\xetex_or_luatex:nnn {
  \define@key[zf]{options}{Mapping}
}{
  \zf@update@family{+map:#1}
  \zf@update@ff{mapping=#1}
}{
  \tl_if_eq:nnTF {#1} {tex-text} {
    \fontspec_warning:n {no-mapping-ligtex}
    \msg_redirect_name:nnn {fontspec} {no-mapping-ligtex} {none}
    \setkeys[zf]{options}{ Ligatures=TeX }
  }{
    \fontspec_warning:n {no-mapping}
  }
}
```

FeatureFile

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

20.9.5 Continuous font axes

```
\define@key[zf]{options}{Weight}{
  \zf@update@family{+weight:#1}
  \zf@update@ff{weight=#1}
}
\define@key[zf]{options}{Width}{
  \zf@update@family{+width:#1}
  \zf@update@ff{width=#1}
}
\define@key[zf]{options}{OpticalSize}{
  \xetex_or_luatex:nn {
    \ifzf@icu
      \tl_set:Nn \l_fontspec_optical_size_tl {/ S = #1}
      \zf@update@family{+size:#1}
    \fi
```

```
    \ifzf@mm
      \zf@update@family{+size:#1}
      \zf@update@ff{optical size=#1}
    \fi
    \ifzf@icu\else
      \ifzf@mm\else
        \ifzf@firsttime
          \fontspec_warning:n {no-opticals}
        \fi
      \fi
    \fi
  }{
    \tl_set:Nn \l_fontspec_optical_size_tl {/ S = #1}
    \zf@update@family{+size:#1}
  }
}
```

20.9.6 Font transformations

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

```
\define@key[zf]{options}{FakeSlant}[0.2]{
  \zf@update@family{+slant:#1}
  \zf@update@ff{slant=#1}
}
\define@key[zf]{options}{FakeStretch}[1.2]{
  \zf@update@family{+extend:#1}
  \zf@update@ff{extend=#1}
}
\define@key[zf]{options}{FakeBold}[1.5]{
  \zf@update@family{+embolden:#1}
  \zf@update@ff{embolden=#1}
}
```

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)

```
\define@key[zf]{options}{AutoFakeSlant}[0.2]{
  \ifzf@firsttime
    \tl_set:Nn \zf@fake@slant {#1}
    \tl_put_right:Nn \zf@it@feat {,FakeSlant=#1}
    \tl_set_eq:NN \zf@it \zf@fontname
    \zf@update@family{fakeit:#1}
    \tl_if_empty:NF \zf@fake@embolden {
      \tl_put_right:Nx \zf@bfit@feat
        {,FakeBold=\zf@fake@embolden,FakeSlant=#1}
      \tl_set_eq:NN \zf@bfit \zf@fontname
    }
  \fi
```

```
}
```

Same but reversed:

```
\define@key[zf]{options}{AutoFakeBold}[1.5]{
  \ifzf@firsttime
    \tl_set:Nn \zf@fake@embolden {#1}
    \tl_put_right:Nn \zf@bf@feat {,FakeBold=#1}
    \tl_set_eq:NN \zf@bf \zf@fontname
    \zf@update@family{fakebf:#1}
    \tl_if_empty:NF \zf@fake@slant {
      \tl_put_right:Nx \zf@bfit@feat
        {,FakeSlant=\zf@fake@slant,FakeBold=#1}
      \tl_set_eq:NN \zf@bfit \zf@fontname
    }
  \fi
}
```

20.9.7 Ligatures

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

```
\zf@define@font@feature{Ligatures}
\zf@define@feature@option{Ligatures}{Required}       {1}{0}{+rlig}
\zf@define@feature@option{Ligatures}{NoRequired}     {1}{1}{-rlig}
\zf@define@feature@option{Ligatures}{Common}         {1}{2}{+liga}
\zf@define@feature@option{Ligatures}{NoCommon}       {1}{3}{-liga}
\zf@define@feature@option{Ligatures}{Rare}           {1}{4}{+dlig}
\zf@define@feature@option{Ligatures}{NoRare}         {1}{5}{-dlig}
\zf@define@feature@option{Ligatures}{Discretionary}  {1}{4}{+dlig}
\zf@define@feature@option{Ligatures}{NoDiscretionary}{1}{5}{-dlig}
\zf@define@feature@option{Ligatures}{Contextual}     {}{}  {+clig}
\zf@define@feature@option{Ligatures}{NoContextual}   {}{}  {-clig}
\zf@define@feature@option{Ligatures}{Historical}     {}{}  {+hlig}
\zf@define@feature@option{Ligatures}{NoHistorical}   {}{}  {-hlig}
\zf@define@feature@option{Ligatures}{Logos}          {1}{6} {}
\zf@define@feature@option{Ligatures}{NoLogos}        {1}{7} {}
\zf@define@feature@option{Ligatures}{Rebus}          {1}{8} {}
\zf@define@feature@option{Ligatures}{NoRebus}        {1}{9} {}
\zf@define@feature@option{Ligatures}{Diphthong}      {1}{10}{}
\zf@define@feature@option{Ligatures}{NoDiphthong}    {1}{11}{}
\zf@define@feature@option{Ligatures}{Squared}        {1}{12}{}
\zf@define@feature@option{Ligatures}{NoSquared}      {1}{13}{}
\zf@define@feature@option{Ligatures}{AbbrevSquared}  {1}{14}{}
\zf@define@feature@option{Ligatures}{NoAbbrevSquared}{1}{15}{}
\zf@define@feature@option{Ligatures}{Icelandic}      {1}{32}{}
\zf@define@feature@option{Ligatures}{NoIcelandic}    {1}{33}{}
```

Emulate CM extra ligatures.

```
\define@key[zf@feat]{Ligatures}{TeX}[]{
  \xetex_or_luatex:nn {
    \zf@update@family{+map:tex-text}
```

```
    \zf@update@ff{mapping=tex-text}
  }{
    \zf@update@family{+tlig+trep}
    \zf@update@ff{+tlig;+trep}
  }
}
```

20.9.8 Letters

```
\zf@define@font@feature{Letters}
\zf@define@feature@option{Letters}{Normal}             {3}{0}{}
\zf@define@feature@option{Letters}{Uppercase}          {3}{1}{+case}
\zf@define@feature@option{Letters}{Lowercase}          {3}{2}{}
\zf@define@feature@option{Letters}{SmallCaps}          {3}{3}{+smcp}
\zf@define@feature@option{Letters}{PetiteCaps}         {} {} {+pcap}
\zf@define@feature@option{Letters}{UppercaseSmallCaps} {} {} {+c2sc}
\zf@define@feature@option{Letters}{UppercasePetiteCaps}{} {} {+c2pc}
\zf@define@feature@option{Letters}{InitialCaps}        {3}{4}{}
\zf@define@feature@option{Letters}{Unicase}            {} {} {+unic}
```

20.9.9 Numbers

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

Both naming conventions are offered to select the number case.

```
\zf@define@font@feature{Numbers}
\zf@define@feature@option{Numbers}{Monospaced}   {6} {0}{+tnum}
\zf@define@feature@option{Numbers}{Proportional} {6} {1}{+pnum}
\zf@define@feature@option{Numbers}{Lowercase}    {21}{0}{+onum}
\zf@define@feature@option{Numbers}{OldStyle}     {21}{0}{+onum}
\zf@define@feature@option{Numbers}{Uppercase}    {21}{1}{+lnum}
\zf@define@feature@option{Numbers}{Lining}       {21}{1}{+lnum}
\zf@define@feature@option{Numbers}{SlashedZero}  {14}{5}{+zero}
\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.

```
\luatex_if_engine:T {
  \zf@define@feature@option{Numbers}{Arabic}{}{}{+anum}
  \zf@define@feature@option{Numbers}{Farsi} {}{}{+anum}
}
```

20.9.10 Contextuals

```
\zf@define@font@feature  {Contextuals}
\zf@define@feature@option{Contextuals}{Swash}        {} {} {+cswh}
\zf@define@feature@option{Contextuals}{NoSwash}      {} {} {-cswh}
\zf@define@feature@option{Contextuals}{Alternate}    {} {} {+calt}
\zf@define@feature@option{Contextuals}{NoAlternate}  {} {} {-calt}
\zf@define@feature@option{Contextuals}{WordInitial}  {8}{0}{+init}
\zf@define@feature@option{Contextuals}{NoWordInitial}{8}{1}{-init}
```

```
\zf@define@feature@option{Contextuals}{WordFinal}    {8}{2}{+fina}
\zf@define@feature@option{Contextuals}{NoWordFinal}  {8}{3}{-fina}
\zf@define@feature@option{Contextuals}{LineInitial}  {8}{4}{}
\zf@define@feature@option{Contextuals}{NoLineInitial}{8}{5}{}
\zf@define@feature@option{Contextuals}{LineFinal}    {8}{6}{+falt}
\zf@define@feature@option{Contextuals}{NoLineFinal}  {8}{7}{-falt}
\zf@define@feature@option{Contextuals}{Inner}        {8}{8}{+medi}
\zf@define@feature@option{Contextuals}{NoInner}      {8}{9}{-medi}
```

20.9.11 Diacritics

```
\zf@define@font@feature{Diacritics}
\zf@define@feature@option{Diacritics}{Show}        {9}{0}{}
\zf@define@feature@option{Diacritics}{Hide}        {9}{1}{}
\zf@define@feature@option{Diacritics}{Decompose}   {9}{2}{}
\zf@define@feature@option{Diacritics}{MarkToBase}  {}{}{+mark}
\zf@define@feature@option{Diacritics}{NoMarkToBase}{}{}{-mark}
\zf@define@feature@option{Diacritics}{MarkToMark}  {}{}{+mkmk}
\zf@define@feature@option{Diacritics}{NoMarkToMark}{}{}{-mkmk}
\zf@define@feature@option{Diacritics}{AboveBase}   {}{}{+abvm}
\zf@define@feature@option{Diacritics}{NoAboveBase} {}{}{-abvm}
\zf@define@feature@option{Diacritics}{BelowBase}   {}{}{+blwm}
\zf@define@feature@option{Diacritics}{NoBelowBase} {}{}{-blwm}
```

20.9.12 Kerning

```
\zf@define@font@feature{Kerning}
\zf@define@feature@option{Kerning}{Uppercase}{}{}{+cpsp}
\zf@define@feature@option{Kerning}{On}       {}{}{+kern}
\zf@define@feature@option{Kerning}{Off}      {}{}{-kern}
%\zf@define@feature@option{Kerning}{Vertical}{}{}{+vkrn}
%\zf@define@feature@option{Kerning}
%    {VerticalAlternateProportional}{}{}{+vpal}
%\zf@define@feature@option{Kerning}{VerticalAlternateHalfWidth}{}{}{+vhal}
```

20.9.13 Vertical position

```
\zf@define@font@feature{VerticalPosition}
\zf@define@feature@option{VerticalPosition}{Normal}     {10}{0}{}
\zf@define@feature@option{VerticalPosition}{Superior}   {10}{1}{+sups}
\zf@define@feature@option{VerticalPosition}{Inferior}   {10}{2}{+subs}
\zf@define@feature@option{VerticalPosition}{Ordinal}    {10}{3}{+ordn}
\zf@define@feature@option{VerticalPosition}{Numerator}  {}  {} {+numr}
\zf@define@feature@option{VerticalPosition}{Denominator}{}  {} {+dnom}
\zf@define@feature@option{VerticalPosition}{ScientificInferior}{}{}{+sinf}
```

20.9.14 Fractions

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

20.9.15 Alternates and variants

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

```
\define@key[zf]{options}{Alternate}[0]{
  \clist_set_eq:NN \l_fontspec_tmpa_clist \XKV@rm
  \setkeys*[zf@feat]{Alternate}{#1}
  \unless\ifx\XKV@rm\@empty
    \def\XKV@tfam{Alternate}
    \fontspec_make_feature:nnn{17}{#1}{+salt=#1}
  \fi
  \clist_set_eq:NN \XKV@rm \l_fontspec_tmpa_clist
}
\define@key[zf]{options}{Variant}{
  \clist_set_eq:NN \l_fontspec_tmpa_clist \XKV@rm
  \setkeys*[zf@feat]{Variant}{#1}
  \unless\ifx\XKV@rm\@empty
    \def\XKV@tfam{Variant}
    \fontspec_make_feature:nnx{18}{#1}{+ss\two@digits{#1}}
  \fi
  \clist_set_eq:NN \XKV@rm \l_fontspec_tmpa_clist
}
\aliasfontfeature{Variant}{StylisticSet}
```

20.9.16 Style

```
\zf@define@font@feature{Style}
\zf@define@feature@option{Style}{Alternate}     {}  {} {+salt}
\zf@define@feature@option{Style}{Italic}        {32}{2}{+ital}
\zf@define@feature@option{Style}{Ruby}          {28}{2}{+ruby}
\zf@define@feature@option{Style}{Swash}         {}  {} {+swsh}
\zf@define@feature@option{Style}{Historic}      {}  {} {+hist}
\zf@define@feature@option{Style}{Display}       {19}{1}{}
\zf@define@feature@option{Style}{Engraved}      {19}{2}{}
\zf@define@feature@option{Style}{TitlingCaps}   {19}{4}{+titl}
\zf@define@feature@option{Style}{TallCaps}      {19}{5}{}
\zf@define@feature@option{Style}{HorizontalKana}{}  {} {+hkna}
\zf@define@feature@option{Style}{VerticalKana}  {}  {} {+vkna}
```

20.9.17 CJK shape

```
\zf@define@font@feature{CJKShape}
\zf@define@feature@option{CJKShape}{Traditional}{20}{0} {+trad}
\zf@define@feature@option{CJKShape}{Simplified} {20}{1} {+smpl}
\zf@define@feature@option{CJKShape}{JIS1978}    {20}{2} {+jp78}
\zf@define@feature@option{CJKShape}{JIS1983}    {20}{3} {+jp83}
\zf@define@feature@option{CJKShape}{JIS1990}    {20}{4} {+jp90}
\zf@define@feature@option{CJKShape}{Expert}     {20}{10}{+expt}
\zf@define@feature@option{CJKShape}{NLC}        {20}{13}{+nlck}
```

20.9.18 Character width

```
\zf@define@font@feature{CharacterWidth}
```

```
\zf@define@feature@option{CharacterWidth}{Proportional}{22}{0}{+pwid}
\zf@define@feature@option{CharacterWidth}{Full}{22}{1}{+fwid}
\zf@define@feature@option{CharacterWidth}{Half}{22}{2}{+hwid}
\zf@define@feature@option{CharacterWidth}{Third}{22}{3}{+twid}
\zf@define@feature@option{CharacterWidth}{Quarter}{22}{4}{+qwid}
\zf@define@feature@option{CharacterWidth}{AlternateProportional}{22}{5}{+palt}
\zf@define@feature@option{CharacterWidth}{AlternateHalf}{22}{6}{+halt}
\zf@define@feature@option{CharacterWidth}{Default}{22}{7}{}
```

20.9.19 Annotation

```
\zf@define@feature@option{Annotation}{Off}{24}{0}{}
\zf@define@feature@option{Annotation}{Box}{24}{1}{}
\zf@define@feature@option{Annotation}{RoundedBox}{24}{2}{}
\zf@define@feature@option{Annotation}{Circle}{24}{3}{}
\zf@define@feature@option{Annotation}{BlackCircle}{24}{4}{}
\zf@define@feature@option{Annotation}{Parenthesis}{24}{5}{}
\zf@define@feature@option{Annotation}{Period}{24}{6}{}
\zf@define@feature@option{Annotation}{RomanNumerals}{24}{7}{}
\zf@define@feature@option{Annotation}{Diamond}{24}{8}{}
\zf@define@feature@option{Annotation}{BlackSquare}{24}{9}{}
\zf@define@feature@option{Annotation}{BlackRoundSquare}{24}{10}{}
\zf@define@feature@option{Annotation}{DoubleCircle}{24}{11}{}
```

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

20.9.20 Vertical

```
\zf@define@font@feature{Vertical}
\define@key[zf@feat]{Vertical}{RotatedGlyphs}[]{
  \ifzf@icu
    \fontspec_make_feature:nnn{}{}{+vrt2}
    \zf@update@family{+vert}
    \zf@update@ff{vertical}
  \else
    \zf@update@family{+vert}
    \zf@update@ff{vertical}
  \fi
}
```

20.9.21 Script

```
\newfontscript{Arabic}{arab}            \newfontscript{Armenian}{armn}
\newfontscript{Balinese}{bali}          \newfontscript{Bengali}{beng}
\newfontscript{Bopomofo}{bopo}          \newfontscript{Braille}{brai}
\newfontscript{Buginese}{bugi}          \newfontscript{Buhid}{buhd}
\newfontscript{Byzantine~Music}{byzm} \newfontscript{Canadian~Syllabics}{cans}
```

```
\newfontscript{Cherokee}{cher}
\newfontscript{CJK~Ideographic}{hani}   \newfontscript{Coptic}{copt}
\newfontscript{Cypriot~Syllabary}{cprt} \newfontscript{Cyrillic}{cyrl}
\newfontscript{Default}{DFLT}           \newfontscript{Deseret}{dsrt}
\newfontscript{Devanagari}{deva}        \newfontscript{Ethiopic}{ethi}
\newfontscript{Georgian}{geor}          \newfontscript{Glagolitic}{glag}
\newfontscript{Gothic}{goth}            \newfontscript{Greek}{grek}
\newfontscript{Gujarati}{gujr}          \newfontscript{Gurmukhi}{guru}
\newfontscript{Hangul~Jamo}{jamo}       \newfontscript{Hangul}{hang}
\newfontscript{Hanunoo}{hano}           \newfontscript{Hebrew}{hebr}
\newfontscript{Hiragana~and~Katakana}{kana}
\newfontscript{Javanese}{java}          \newfontscript{Kannada}{knda}
\newfontscript{Kharosthi}{khar}         \newfontscript{Khmer}{khmr}
\newfontscript{Lao}{lao~}               \newfontscript{Latin}{latn}
\newfontscript{Limbu}{limb}             \newfontscript{Linear~B}{linb}
\newfontscript{Malayalam}{mlym}         \newfontscript{Math}{math}
\newfontscript{Mongolian}{mong}
\newfontscript{Musical~Symbols}{musc}   \newfontscript{Myanmar}{mymr}
\newfontscript{N'ko}{nko~}              \newfontscript{Ogham}{ogam}
\newfontscript{Old~Italic}{ital}
\newfontscript{Old~Persian~Cuneiform}{xpeo}
\newfontscript{Oriya}{orya}             \newfontscript{Osmanya}{osma}
\newfontscript{Phags-pa}{phag}          \newfontscript{Phoenician}{phnx}
\newfontscript{Runic}{runr}             \newfontscript{Shavian}{shaw}
\newfontscript{Sinhala}{sinh}
\newfontscript{Sumero-Akkadian~Cuneiform}{xsux}
\newfontscript{Syloti~Nagri}{sylo}      \newfontscript{Syriac}{syrc}
\newfontscript{Tagalog}{tglg}           \newfontscript{Tagbanwa}{tagb}
\newfontscript{Tai~Le}{tale}            \newfontscript{Tai~Lu}{talu}
\newfontscript{Tamil}{taml}             \newfontscript{Telugu}{telu}
\newfontscript{Thaana}{thaa}            \newfontscript{Thai}{thai}
\newfontscript{Tibetan}{tibt}           \newfontscript{Tifinagh}{tfng}
\newfontscript{Ugaritic~Cuneiform}{ugar}\newfontscript{Yi}{yi~~}
```

For convenience:

```
\newfontscript{Kana}{kana}
\newfontscript{Maths}{math}
\newfontscript{CJK}{hani}
```

20.9.22 Language

```
\newfontlanguage{Abaza}{ABA}\newfontlanguage{Abkhazian}{ABK}
\newfontlanguage{Adyghe}{ADY}\newfontlanguage{Afrikaans}{AFK}
\newfontlanguage{Afar}{AFR}\newfontlanguage{Agaw}{AGW}
\newfontlanguage{Altai}{ALT}\newfontlanguage{Amharic}{AMH}
\newfontlanguage{Arabic}{ARA}\newfontlanguage{Aari}{ARI}
\newfontlanguage{Arakanese}{ARK}\newfontlanguage{Assamese}{ASM}
\newfontlanguage{Athapaskan}{ATH}\newfontlanguage{Avar}{AVR}
\newfontlanguage{Awadhi}{AWA}\newfontlanguage{Aymara}{AYM}
\newfontlanguage{Azeri}{AZE}\newfontlanguage{Badaga}{BAD}
\newfontlanguage{Baghelkhandi}{BAG}\newfontlanguage{Balkar}{BAL}
\newfontlanguage{Baule}{BAU}\newfontlanguage{Berber}{BBR}
\newfontlanguage{Bench}{BCH}\newfontlanguage{Bible~Cree}{BCR}
```

```
\newfontlanguage{Belarussian}{BEL}\newfontlanguage{Bemba}{BEM}
\newfontlanguage{Bengali}{BEN}\newfontlanguage{Bulgarian}{BGR}
\newfontlanguage{Bhili}{BHI}\newfontlanguage{Bhojpuri}{BHO}
\newfontlanguage{Bikol}{BIK}\newfontlanguage{Bilen}{BIL}
\newfontlanguage{Blackfoot}{BKF}\newfontlanguage{Balochi}{BLI}
\newfontlanguage{Balante}{BLN}\newfontlanguage{Balti}{BLT}
\newfontlanguage{Bambara}{BMB}\newfontlanguage{Bamileke}{BML}
\newfontlanguage{Breton}{BRE}\newfontlanguage{Brahui}{BRH}
\newfontlanguage{Braj~Bhasha}{BRI}\newfontlanguage{Burmese}{BRM}
\newfontlanguage{Bashkir}{BSH}\newfontlanguage{Beti}{BTI}
\newfontlanguage{Catalan}{CAT}\newfontlanguage{Cebuano}{CEB}
\newfontlanguage{Chechen}{CHE}\newfontlanguage{Chaha~Gurage}{CHG}
\newfontlanguage{Chattisgarhi}{CHH}\newfontlanguage{Chichewa}{CHI}
\newfontlanguage{Chukchi}{CHK}\newfontlanguage{Chipewyan}{CHP}
\newfontlanguage{Cherokee}{CHR}\newfontlanguage{Chuvash}{CHU}
\newfontlanguage{Comorian}{CMR}\newfontlanguage{Coptic}{COP}
\newfontlanguage{Cree}{CRE}\newfontlanguage{Carrier}{CRR}
\newfontlanguage{Crimean~Tatar}{CRT}\newfontlanguage{Church~Slavonic}{CSL}
\newfontlanguage{Czech}{CSY}\newfontlanguage{Danish}{DAN}
\newfontlanguage{Dargwa}{DAR}\newfontlanguage{Woods~Cree}{DCR}
\newfontlanguage{German}{DEU}
\newfontlanguage{Dogri}{DGR}\newfontlanguage{Divehi}{DIV}
\newfontlanguage{Djerma}{DJR}\newfontlanguage{Dangme}{DNG}
\newfontlanguage{Dinka}{DNK}\newfontlanguage{Dungan}{DUN}
\newfontlanguage{Dzongkha}{DZN}\newfontlanguage{Ebira}{EBI}
\newfontlanguage{Eastern~Cree}{ECR}\newfontlanguage{Edo}{EDO}
\newfontlanguage{Efik}{EFI}\newfontlanguage{Greek}{ELL}
\newfontlanguage{English}{ENG}\newfontlanguage{Erzya}{ERZ}
\newfontlanguage{Spanish}{ESP}\newfontlanguage{Estonian}{ETI}
\newfontlanguage{Basque}{EUQ}\newfontlanguage{Evenki}{EVK}
\newfontlanguage{Even}{EVN}\newfontlanguage{Ewe}{EWE}
\newfontlanguage{French~Antillean}{FAN}\newfontlanguage{Farsi}{FAR}
\newfontlanguage{Finnish}{FIN}\newfontlanguage{Fijian}{FJI}
\newfontlanguage{Flemish}{FLE}\newfontlanguage{Forest~Nenets}{FNE}
\newfontlanguage{Fon}{FON}\newfontlanguage{Faroese}{FOS}
\newfontlanguage{French}{FRA}\newfontlanguage{Frisian}{FRI}
\newfontlanguage{Friulian}{FRL}\newfontlanguage{Futa}{FTA}
\newfontlanguage{Fulani}{FUL}\newfontlanguage{Ga}{GAD}
\newfontlanguage{Gaelic}{GAE}\newfontlanguage{Gagauz}{GAG}
\newfontlanguage{Galician}{GAL}\newfontlanguage{Garshuni}{GAR}
\newfontlanguage{Garhwali}{GAW}\newfontlanguage{Ge'ez}{GEZ}
\newfontlanguage{Gilyak}{GIL}\newfontlanguage{Gumuz}{GMZ}
\newfontlanguage{Gondi}{GON}\newfontlanguage{Greenlandic}{GRN}
\newfontlanguage{Garo}{GRO}\newfontlanguage{Guarani}{GUA}
\newfontlanguage{Gujarati}{GUJ}\newfontlanguage{Haitian}{HAI}
\newfontlanguage{Halam}{HAL}\newfontlanguage{Harauti}{HAR}
\newfontlanguage{Hausa}{HAU}\newfontlanguage{Hawaiin}{HAW}
\newfontlanguage{Hammer-Banna}{HBN}\newfontlanguage{Hiligaynon}{HIL}
\newfontlanguage{Hindi}{HIN}\newfontlanguage{High~Mari}{HMA}
\newfontlanguage{Hindko}{HND}\newfontlanguage{Ho}{HO}
\newfontlanguage{Harari}{HRI}\newfontlanguage{Croatian}{HRV}
```

```
\newfontlanguage{Hungarian}{HUN}\newfontlanguage{Armenian}{HYE}
\newfontlanguage{Igbo}{IBO}\newfontlanguage{Ijo}{IJO}
\newfontlanguage{Ilokano}{ILO}\newfontlanguage{Indonesian}{IND}
\newfontlanguage{Ingush}{ING}\newfontlanguage{Inuktitut}{INU}
\newfontlanguage{Irish}{IRI}\newfontlanguage{Irish~Traditional}{IRT}
\newfontlanguage{Icelandic}{ISL}\newfontlanguage{Inari~Sami}{ISM}
\newfontlanguage{Italian}{ITA}\newfontlanguage{Hebrew}{IWR}
\newfontlanguage{Javanese}{JAV}\newfontlanguage{Yiddish}{JII}
\newfontlanguage{Japanese}{JAN}\newfontlanguage{Judezmo}{JUD}
\newfontlanguage{Jula}{JUL}\newfontlanguage{Kabardian}{KAB}
\newfontlanguage{Kachchi}{KAC}\newfontlanguage{Kalenjin}{KAL}
\newfontlanguage{Kannada}{KAN}\newfontlanguage{Karachay}{KAR}
\newfontlanguage{Georgian}{KAT}\newfontlanguage{Kazakh}{KAZ}
\newfontlanguage{Kebena}{KEB}\newfontlanguage{Khutsuri~Georgian}{KGE}
\newfontlanguage{Khakass}{KHA}\newfontlanguage{Khanty-Kazim}{KHK}
\newfontlanguage{Khmer}{KHM}\newfontlanguage{Khanty-Shurishkar}{KHS}
\newfontlanguage{Khanty-Vakhi}{KHV}\newfontlanguage{Khowar}{KHW}
\newfontlanguage{Kikuyu}{KIK}\newfontlanguage{Kirghiz}{KIR}
\newfontlanguage{Kisii}{KIS}\newfontlanguage{Kokni}{KKN}
\newfontlanguage{Kalmyk}{KLM}\newfontlanguage{Kamba}{KMB}
\newfontlanguage{Kumaoni}{KMN}\newfontlanguage{Komo}{KMO}
\newfontlanguage{Komso}{KMS}\newfontlanguage{Kanuri}{KNR}
\newfontlanguage{Kodagu}{KOD}\newfontlanguage{Korean~Old~Hangul}{KOH}
\newfontlanguage{Konkani}{KOK}\newfontlanguage{Kikongo}{KON}
\newfontlanguage{Komi-Permyak}{KOP}\newfontlanguage{Korean}{KOR}
\newfontlanguage{Komi-Zyrian}{KOZ}\newfontlanguage{Kpelle}{KPL}
\newfontlanguage{Krio}{KRI}\newfontlanguage{Karakalpak}{KRK}
\newfontlanguage{Karelian}{KRL}\newfontlanguage{Karaim}{KRM}
\newfontlanguage{Karen}{KRN}\newfontlanguage{Koorete}{KRT}
\newfontlanguage{Kashmiri}{KSH}\newfontlanguage{Khasi}{KSI}
\newfontlanguage{Kildin~Sami}{KSM}\newfontlanguage{Kui}{KUI}
\newfontlanguage{Kulvi}{KUL}\newfontlanguage{Kumyk}{KUM}
\newfontlanguage{Kurdish}{KUR}\newfontlanguage{Kurukh}{KUU}
\newfontlanguage{Kuy}{KUY}\newfontlanguage{Koryak}{KYK}
\newfontlanguage{Ladin}{LAD}\newfontlanguage{Lahuli}{LAH}
\newfontlanguage{Lak}{LAK}\newfontlanguage{Lambani}{LAM}
\newfontlanguage{Lao}{LAO}\newfontlanguage{Latin}{LAT}
\newfontlanguage{Laz}{LAZ}\newfontlanguage{L-Cree}{LCR}
\newfontlanguage{Ladakhi}{LDK}\newfontlanguage{Lezgi}{LEZ}
\newfontlanguage{Lingala}{LIN}\newfontlanguage{Low~Mari}{LMA}
\newfontlanguage{Limbu}{LMB}\newfontlanguage{Lomwe}{LMW}
\newfontlanguage{Lower~Sorbian}{LSB}\newfontlanguage{Lule~Sami}{LSM}
\newfontlanguage{Lithuanian}{LTH}\newfontlanguage{Luba}{LUB}
\newfontlanguage{Luganda}{LUG}\newfontlanguage{Luhya}{LUH}
\newfontlanguage{Luo}{LUO}\newfontlanguage{Latvian}{LVI}
\newfontlanguage{Majang}{MAJ}\newfontlanguage{Makua}{MAK}
\newfontlanguage{Malayalam~Traditional}{MAL}\newfontlanguage{Mansi}{MAN}
\newfontlanguage{Marathi}{MAR}\newfontlanguage{Marwari}{MAW}
\newfontlanguage{Mbundu}{MBN}\newfontlanguage{Manchu}{MCH}
\newfontlanguage{Moose~Cree}{MCR}\newfontlanguage{Mende}{MDE}
\newfontlanguage{Me'en}{MEN}\newfontlanguage{Mizo}{MIZ}
```

```
\newfontlanguage{Macedonian}{MKD}\newfontlanguage{Male}{MLE}
\newfontlanguage{Malagasy}{MLG}\newfontlanguage{Malinke}{MLN}
\newfontlanguage{Malayalam~Reformed}{MLR}\newfontlanguage{Malay}{MLY}
\newfontlanguage{Mandinka}{MND}\newfontlanguage{Mongolian}{MNG}
\newfontlanguage{Manipuri}{MNI}\newfontlanguage{Maninka}{MNK}
\newfontlanguage{Manx~Gaelic}{MNX}\newfontlanguage{Moksha}{MOK}
\newfontlanguage{Moldavian}{MOL}\newfontlanguage{Mon}{MON}
\newfontlanguage{Moroccan}{MOR}\newfontlanguage{Maori}{MRI}
\newfontlanguage{Maithili}{MTH}\newfontlanguage{Maltese}{MTS}
\newfontlanguage{Mundari}{MUN}\newfontlanguage{Naga-Assamese}{NAG}
\newfontlanguage{Nanai}{NAN}\newfontlanguage{Naskapi}{NAS}
\newfontlanguage{N-Cree}{NCR}\newfontlanguage{Ndebele}{NDB}
\newfontlanguage{Ndonga}{NDG}\newfontlanguage{Nepali}{NEP}
\newfontlanguage{Newari}{NEW}\newfontlanguage{Nagari}{NGR}
\newfontlanguage{Norway~House~Cree}{NHC}\newfontlanguage{Nisi}{NIS}
\newfontlanguage{Niuean}{NIU}\newfontlanguage{Nkole}{NKL}
\newfontlanguage{N'ko}{NKO}\newfontlanguage{Dutch}{NLD}
\newfontlanguage{Nogai}{NOG}\newfontlanguage{Norwegian}{NOR}
\newfontlanguage{Northern~Sami}{NSM}\newfontlanguage{Northern~Tai}{NTA}
\newfontlanguage{Esperanto}{NTO}\newfontlanguage{Nynorsk}{NYN}
\newfontlanguage{Oji-Cree}{OCR}\newfontlanguage{Ojibway}{OJB}
\newfontlanguage{Oriya}{ORI}\newfontlanguage{Oromo}{ORO}
\newfontlanguage{Ossetian}{OSS}\newfontlanguage{Palestinian~Aramaic}{PAA}
\newfontlanguage{Pali}{PAL}\newfontlanguage{Punjabi}{PAN}
\newfontlanguage{Palpa}{PAP}\newfontlanguage{Pashto}{PAS}
\newfontlanguage{Polytonic~Greek}{PGR}\newfontlanguage{Pilipino}{PIL}
\newfontlanguage{Palaung}{PLG}\newfontlanguage{Polish}{PLK}
\newfontlanguage{Provencal}{PRO}\newfontlanguage{Portuguese}{PTG}
\newfontlanguage{Chin}{QIN}\newfontlanguage{Rajasthani}{RAJ}
\newfontlanguage{R-Cree}{RCR}\newfontlanguage{Russian~Buriat}{RBU}
\newfontlanguage{Riang}{RIA}\newfontlanguage{Rhaeto-Romanic}{RMS}
\newfontlanguage{Romanian}{ROM}\newfontlanguage{Romany}{ROY}
\newfontlanguage{Rusyn}{RSY}\newfontlanguage{Ruanda}{RUA}
\newfontlanguage{Russian}{RUS}\newfontlanguage{Sadri}{SAD}
\newfontlanguage{Sanskrit}{SAN}\newfontlanguage{Santali}{SAT}
\newfontlanguage{Sayisi}{SAY}\newfontlanguage{Sekota}{SEK}
\newfontlanguage{Selkup}{SEL}\newfontlanguage{Sango}{SGO}
\newfontlanguage{Shan}{SHN}\newfontlanguage{Sibe}{SIB}
\newfontlanguage{Sidamo}{SID}\newfontlanguage{Silte~Gurage}{SIG}
\newfontlanguage{Skolt~Sami}{SKS}\newfontlanguage{Slovak}{SKY}
\newfontlanguage{Slavey}{SLA}\newfontlanguage{Slovenian}{SLV}
\newfontlanguage{Somali}{SML}\newfontlanguage{Samoan}{SMO}
\newfontlanguage{Sena}{SNA}\newfontlanguage{Sindhi}{SND}
\newfontlanguage{Sinhalese}{SNH}\newfontlanguage{Soninke}{SNK}
\newfontlanguage{Sodo~Gurage}{SOG}\newfontlanguage{Sotho}{SOT}
\newfontlanguage{Albanian}{SQI}\newfontlanguage{Serbian}{SRB}
\newfontlanguage{Saraiki}{SRK}\newfontlanguage{Serer}{SRR}
\newfontlanguage{South~Slavey}{SSL}\newfontlanguage{Southern~Sami}{SSM}
\newfontlanguage{Suri}{SUR}\newfontlanguage{Svan}{SVA}
\newfontlanguage{Swedish}{SVE}\newfontlanguage{Swadaya~Aramaic}{SWA}
\newfontlanguage{Swahili}{SWK}\newfontlanguage{Swazi}{SWZ}
```

```
\newfontlanguage{Sutu}{SXT}\newfontlanguage{Syriac}{SYR}
\newfontlanguage{Tabasaran}{TAB}\newfontlanguage{Tajiki}{TAJ}
\newfontlanguage{Tamil}{TAM}\newfontlanguage{Tatar}{TAT}
\newfontlanguage{TH-Cree}{TCR}\newfontlanguage{Telugu}{TEL}
\newfontlanguage{Tongan}{TGN}\newfontlanguage{Tigre}{TGR}
\newfontlanguage{Tigrinya}{TGY}\newfontlanguage{Thai}{THA}
\newfontlanguage{Tahitian}{THT}\newfontlanguage{Tibetan}{TIB}
\newfontlanguage{Turkmen}{TKM}\newfontlanguage{Temne}{TMN}
\newfontlanguage{Tswana}{TNA}\newfontlanguage{Tundra~Nenets}{TNE}
\newfontlanguage{Tonga}{TNG}\newfontlanguage{Todo}{TOD}
\newfontlanguage{Tsonga}{TSG}\newfontlanguage{Turoyo~Aramaic}{TUA}
\newfontlanguage{Tulu}{TUL}\newfontlanguage{Tuvin}{TUV}
\newfontlanguage{Twi}{TWI}\newfontlanguage{Udmurt}{UDM}
\newfontlanguage{Ukrainian}{UKR}\newfontlanguage{Urdu}{URD}
\newfontlanguage{Upper~Sorbian}{USB}\newfontlanguage{Uyghur}{UYG}
\newfontlanguage{Uzbek}{UZB}\newfontlanguage{Venda}{VEN}
\newfontlanguage{Vietnamese}{VIT}\newfontlanguage{Wa}{WA}
\newfontlanguage{Wagdi}{WAG}\newfontlanguage{West-Cree}{WCR}
\newfontlanguage{Welsh}{WEL}\newfontlanguage{Wolof}{WLF}
\newfontlanguage{Tai~Lue}{XBD}\newfontlanguage{Xhosa}{XHS}
\newfontlanguage{Yakut}{YAK}\newfontlanguage{Yoruba}{YBA}
\newfontlanguage{Y-Cree}{YCR}\newfontlanguage{Yi~Classic}{YIC}
\newfontlanguage{Yi~Modern}{YIM}\newfontlanguage{Chinese~Hong~Kong}{ZHH}
\newfontlanguage{Chinese~Phonetic}{ZHP}\newfontlanguage{Chinese~Simplified}{ZHS}
\newfontlanguage{Chinese~Traditional}{ZHT}\newfontlanguage{Zande}{ZND}
\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:

```
\define@key[zf@feat]{Lang}{Turkish}[]{
  \fontspec_check_lang:nTF {TRK} {
    \c@zf@language\l_fontspec_strnum_int\relax
    \zf@update@family{+lang=Turkish}
    \tl_set:Nn \l_fontspec_lang_tl {TRK}
  }{
    \fontspec_check_lang:nTF {TUR} {
      \c@zf@language\l_fontspec_strnum_int\relax
      \zf@update@family{+lang=Turkish}
      \tl_set:Nn \l_fontspec_lang_tl {TUR}
    }{
      \fontspec_warning:nx {language-not-exist} {#1}
    }
  }
}
```

Default

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

20.9.23 Raw feature string

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

```
\define@key[zf]{options}{RawFeature}{
  \zf@update@family{+Raw:#1}
  \zf@update@ff{#1}
}
```

20.10 Italic small caps

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

`\sishape` `\textsi` First, the commands for actually selecting italic small caps are defined. I use `si` as the NFSS shape for italic small caps, but I have seen `itsc` and `slsc` also used. `\sidefault` may be redefined to one of these if required for compatibility.

```
\providecommand*{\sidefault}{si}
\DeclareRobustCommand{\sishape}{
  \not@math@alphabet\sishape\relax
  \fontshape\sidefault\selectfont
}
\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.

```
\newcommand*{\zf@merge@shape}[3]{
  \edef\@tempa{#1}
  \edef\@tempb{#2}
  \ifx\f@shape\@tempb
    \ifcsname\f@encoding/\f@family/\f@series/#3\endcsname
      \edef\@tempa{#3}
    \fi
  \fi
  \fontshape{\@tempa}\selectfont
}
```

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

```
\DeclareRobustCommand \itshape {
  \not@math@alphabet\itshape\mathit
  \zf@merge@shape\itdefault\scdefault\sidefault
}
\DeclareRobustCommand \slshape {
  \not@math@alphabet\slshape\relax
  \zf@merge@shape\sldefault\scdefault\sidefault
```

```
}
\DeclareRobustCommand \scshape {
  \not@math@alphabet\scshape\relax
  \zf@merge@shape\scdefault\itdefault\sidefault
}
\DeclareRobustCommand \upshape {
  \not@math@alphabet\upshape\relax
  \zf@merge@shape\updefault\sidefault\scdefault
}
```

20.11 Selecting maths fonts

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

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

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

```
\@ifpackageloaded{euler}{\zf@package@euler@loadedtrue}
                        {\zf@package@euler@loadedfalse}
\def\zf@math{
  \let\zf@font@warning\@font@warning
  \let\@font@warning\@font@info
  \@ifpackageloaded{euler}{
    \ifzf@package@euler@loaded
      \zf@math@eulertrue
    \else
      \fontspec_error:n {euler-too-late}
    \fi
  }{}
  \@ifpackageloaded{lucbmath}{\zf@math@lucidatrue}{}
  \@ifpackageloaded{lucidabr}{\zf@math@lucidatrue}{}
  \@ifpackageloaded{lucimatx}{\zf@math@lucidatrue}{}
```

Knuth's CM fonts 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 LaTeX'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 `EulerFractur`, but it's *ugly*. So I ignore it. Sorry if this causes inconvenience.)

```
  \DeclareSymbolFont{legacymaths}{OT1}{cmr}{m}{n}
  \SetSymbolFont{legacymaths}{bold}{OT1}{cmr}{bx}{n}
```

```
\DeclareMathAccent{\acute}   {\mathalpha}{legacymaths}{19}
\DeclareMathAccent{\grave}   {\mathalpha}{legacymaths}{18}
\DeclareMathAccent{\ddot}    {\mathalpha}{legacymaths}{127}
\DeclareMathAccent{\tilde}   {\mathalpha}{legacymaths}{126}
\DeclareMathAccent{\bar}     {\mathalpha}{legacymaths}{22}
\DeclareMathAccent{\breve}   {\mathalpha}{legacymaths}{21}
\DeclareMathAccent{\check}   {\mathalpha}{legacymaths}{20}
\DeclareMathAccent{\hat}     {\mathalpha}{legacymaths}{94} % too bad, euler
\DeclareMathAccent{\dot}     {\mathalpha}{legacymaths}{95}
\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{:}\mskip6muplus1mu\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.

```
\begingroup
  \mathchardef\@tempa="603A \relax
  \let\next\egroup
  \ifx\colon\@tempa
    \DeclareMathSymbol{\colon}{\mathpunct}{legacymaths}{58}
  \fi
\endgroup
```

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

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

```
  \ifzf@math@lucida\else
    \DeclareMathSymbol{0}{\mathalpha}{legacymaths}{‘0}
    \DeclareMathSymbol{1}{\mathalpha}{legacymaths}{‘1}
    \DeclareMathSymbol{2}{\mathalpha}{legacymaths}{‘2}
    \DeclareMathSymbol{3}{\mathalpha}{legacymaths}{‘3}
    \DeclareMathSymbol{4}{\mathalpha}{legacymaths}{‘4}
    \DeclareMathSymbol{5}{\mathalpha}{legacymaths}{‘5}
    \DeclareMathSymbol{6}{\mathalpha}{legacymaths}{‘6}
    \DeclareMathSymbol{7}{\mathalpha}{legacymaths}{‘7}
    \DeclareMathSymbol{8}{\mathalpha}{legacymaths}{‘8}
    \DeclareMathSymbol{9}{\mathalpha}{legacymaths}{‘9}
    \DeclareMathSymbol{\Gamma}{\mathalpha}{legacymaths}{0}
    \DeclareMathSymbol{\Delta}{\mathalpha}{legacymaths}{1}
    \DeclareMathSymbol{\Theta}{\mathalpha}{legacymaths}{2}
    \DeclareMathSymbol{\Lambda}{\mathalpha}{legacymaths}{3}
    \DeclareMathSymbol{\Xi}{\mathalpha}{legacymaths}{4}
    \DeclareMathSymbol{\Pi}{\mathalpha}{legacymaths}{5}
    \DeclareMathSymbol{\Sigma}{\mathalpha}{legacymaths}{6}
    \DeclareMathSymbol{\Upsilon}{\mathalpha}{legacymaths}{7}
    \DeclareMathSymbol{\Phi}{\mathalpha}{legacymaths}{8}
    \DeclareMathSymbol{\Psi}{\mathalpha}{legacymaths}{9}
    \DeclareMathSymbol{\Omega}{\mathalpha}{legacymaths}{10}
    \DeclareMathSymbol{+}{\mathbin}{legacymaths}{43}
    \DeclareMathSymbol{=}{\mathrel}{legacymaths}{61}
    \DeclareMathDelimiter{(}{\mathopen} {legacymaths}{40}{largesymbols}{0}
    \DeclareMathDelimiter{)}{\mathclose}{legacymaths}{41}{largesymbols}{1}
    \DeclareMathDelimiter{[}{\mathopen} {legacymaths}{91}{largesymbols}{2}
    \DeclareMathDelimiter{]}{\mathclose}{legacymaths}{93}{largesymbols}{3}
    \DeclareMathDelimiter{/}{\mathord}{legacymaths}{47}{largesymbols}{14}
    \DeclareMathSymbol{\mathdollar}{\mathord}{legacymaths}{36}
  \fi
\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`.

```
\DeclareSymbolFont{operators}\zf@enc\zf@rmmaths\mddefault\updefault
\SetSymbolFont{operators}{normal}\zf@enc\zf@rmmaths\mddefault\updefault
\SetMathAlphabet\mathrm{normal}\zf@enc\zf@rmmaths\mddefault\updefault
\SetMathAlphabet\mathit{normal}\zf@enc\zf@rmmaths\mddefault\itdefault
\SetMathAlphabet\mathbf{normal}\zf@enc\zf@rmmaths\bfdefault\updefault
\SetMathAlphabet\mathsf{normal}\zf@enc\zf@sfmaths\mddefault\updefault
\SetMathAlphabet\mathtt{normal}\zf@enc\zf@ttmaths\mddefault\updefault
\SetSymbolFont{operators}{bold}\zf@enc\zf@rmmaths\bfdefault\updefault
\ifdefined\zf@rmboldmaths
  \SetMathAlphabet\mathrm{bold}\zf@enc\zf@rmboldmaths\mddefault\updefault
```

```
    \SetMathAlphabet\mathbf{bold}\zf@enc\zf@rmboldmaths\bfdefault\updefault
    \SetMathAlphabet\mathit{bold}\zf@enc\zf@rmboldmaths\mddefault\itdefault
  \else
    \SetMathAlphabet\mathrm{bold}\zf@enc\zf@rmmaths\bfdefault\updefault
    \SetMathAlphabet\mathit{bold}\zf@enc\zf@rmmaths\bfdefault\itdefault
  \fi
  \SetMathAlphabet\mathsf{bold}\zf@enc\zf@sfmaths\bfdefault\updefault
  \SetMathAlphabet\mathtt{bold}\zf@enc\zf@ttmaths\bfdefault\updefault
  \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 'LaTeXFont 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.

```
\def\zf@math@maybe{
  \@ifpackageloaded{anttor}{
    \ifx\define@antt@mathversions a\@zf@mathfalse\fi}{}
  \@ifpackageloaded{arev}{\@zf@mathfalse}{}
  \@ifpackageloaded{eulervm}{\@zf@mathfalse}{}
  \@ifpackageloaded{mathdesign}{\@zf@mathfalse}{}
  \@ifpackageloaded{concmath}{\@zf@mathfalse}{}
  \@ifpackageloaded{cmbright}{\@zf@mathfalse}{}
  \@ifpackageloaded{mathesf}{\@zf@mathfalse}{}
  \@ifpackageloaded{gfsartemisia}{\@zf@mathfalse}{}
  \@ifpackageloaded{gfsneohellenic}{\@zf@mathfalse}{}
  \@ifpackageloaded{iwona}{
    \ifx\define@iwona@mathversions a\@zf@mathfalse\fi}{}
  \@ifpackageloaded{kpfonts}{\@zf@mathfalse}{}
  \@ifpackageloaded{kmath}{\@zf@mathfalse}{}
  \@ifpackageloaded{kurier}{
    \ifx\define@kurier@mathversions a\@zf@mathfalse\fi}{}
  \@ifpackageloaded{fouriernc}{\@zf@mathfalse}{}
  \@ifpackageloaded{fourier}{\@zf@mathfalse}{}
  \@ifpackageloaded{mathpazo}{\@zf@mathfalse}{}
  \@ifpackageloaded{mathptmx}{\@zf@mathfalse}{}
  \@ifpackageloaded{MinionPro}{\@zf@mathfalse}{}
  \@ifpackageloaded{unicode-math}{\@zf@mathfalse}{}
  \@ifpackageloaded{breqn}{\@zf@mathfalse}{}
  \if@zf@math
    \fontspec_info:n {setup-math}
    \zf@math
  \fi
}
\AtBeginDocument{\zf@math@maybe}
```

20.12 Finishing up

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

```
\if@zf@configfile
```

```
\InputIfFileExists{fontspec.cfg}
  {\typeout{fontspec.cfg~ loaded.}}
  {\typeout{No~ fontspec.cfg~ file~ found;~ no~ configuration~ loaded.}}
\fi
```

The end! Thanks for coming.

Part VII
fontspec.lua

First we define some metadata.

```
fontspec          = { }

fontspec.module   = {
    name          = "fontspec",
    version       = 2.0,
    date          = "2009/12/04",
    description   = "Advanced font selection for LuaLaTeX.",
    author        = "Khaled Hosny",
    copyright     = "Khaled Hosny",
    license       = "LPPL"
}

luatexbase.provides_module(fontspec.module)

```

Some utility functions

```

utf = unicode.utf8

function fontspec.log    (...) luatexbase.module_log    (fontspec.module.name, string.format(...))
function fontspec.warning(...) luatexbase.module_warning(fontspec.module.name, string.format(...))
function fontspec.error  (...) luatexbase.module_error  (fontspec.module.name, string.format(...))

function fontspec.sprint (...) tex.sprint(luatexbase.catcodetables['latex-package'], ...) end

```

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

```

local function check_script(id, script)
    local s = string.lower(script)
    if id and id  0 then
        local otfdata = fonts.ids[id].shared.otfdata
        if otfdata then
            local features = otfdata.luatex.features
            for i,_ in pairs(features) do
                for j,_ in pairs(features[i]) do
                    if features[i][j][s] then
                        fontspec.log("script '%s' exists in font '%s'",
                                     script, fonts.ids[id].fullname)
                        return true
                    end
                end
            end
        end
    end
```

```
end

local function check_language(id, language, script)
    local s = string.lower(script)
    local l = string.lower(language)
    if id and id  0 then
        local otfdata = fonts.ids[id].shared.otfdata
        if otfdata then
            local features = otfdata.luatex.features
            for i,_ in pairs(features) do
                for j,_ in pairs(features[i]) do
                    if features[i][j][s] and features[i][j][s][l] then
                        fontspec.log("language '%s' for script '%s' exists in font '%s'",
                                     language, script, fonts.ids[id].fullname)
                        return true
                    end
                end
            end
        end
    end
end

local function check_feature(id, feature, language, script)
    local s = string.lower(script)
    local l = string.lower(language)
    local f = string.lower(feature:gsub("^[+-]", ""))
    if id and id  0 then
        local otfdata = fonts.ids[id].shared.otfdata
        if otfdata then
            local features = otfdata.luatex.features
            for i,_ in pairs(features) do
                if features[i][f] and features[i][f][s] then
                    if features[i][f][s][l] == true then
                        fontspec.log("feature '%s' for language '%s' and script '%s' exists in fon
                                     feature, language, script, fonts.ids[id].fullname)
                        return true
                    end
                end
            end
        end
    end
end

```

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

```

local function font_id(str)
    local id
    if tex.luatexversion = 47 then
        id = font.id(str)
    else
```

```
        id = token.create(str)[2]
    end
    return id
end
```

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

```

local function tempswatrue()  fontspec.sprint([[\@tempswatrue]])  end
local function tempswafalse() fontspec.sprint([[\@tempswafalse]]) end

function fontspec.check_ot_script(fnt, script)
    if check_script(font_id(fnt), script) then
        tempswatrue()
    else
        tempswafalse()
    end
end

function fontspec.check_ot_lang(fnt, lang, script)
    if check_language(font_id(fnt), lang, script) then
        tempswatrue()
    else
        tempswafalse()
    end
end

function fontspec.check_ot_feat(fnt, feat, lang, script)
    for _, f in ipairs { "+trep", "+tlig", "+anum" } do
        if feat == f then
            tempswatrue()
            return
        end
    end
    if check_feature(font_id(fnt), feat, lang, script) then
        tempswatrue()
    else
        tempswafalse()
    end
end

function fontspec.charglyph(char)
    if char then
        local id, c
        if utf.len(char)  1 then
            c = utf.byte(utf.char(char:gsub('"', '0x')))
        else
            c = utf.byte(char)
        end

        id = font.current()

```

```
        if font.fonts[id]["characters"][c] then
            return font.fonts[id]["characters"][c].index
        else
            return 0
        end
    else
        return 0
    end
end

```

Part VIII
fontspec-patches.sty

```
\ExplSyntaxOn
```

20.13 Unicode footnote symbols

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

20.14 Emph

\em
\emph
\emshape
\eminnershape

Redefinition of {\em ...} and \emph{...} to use NFSS info to detect when the inner shape should be used.

```
\DeclareRobustCommand \em {
  \@nomath\em
  \tl_if_eq:xxTF \f@shape \itdefault \eminnershape \emshape
}
\DeclareTextFontCommand{\emph}{\em}
\let\emshape\itshape
\let\eminnershape\upshape
```

20.15 \-

\-

This macro is courtesy of Frank Mittelbach and the LaTeX 2_ε source code.

```
\DeclareRobustCommand{\-}{%
  \discretionary{%
    \char\ifnum\hyphenchar\font\z@
           \xlx@defaulthyphenchar
         \else
           \hyphenchar\font
         \fi}{}{}}
\def\xlx@defaulthyphenchar{`\-}
```

20.16 Verbatims

Many thanks to Apostolos Syropoulos for discovering this problem and writing the redefinion of LaTeX's `verbatim` environment and `\verb*` command.

\xxt@visiblespace

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

```
\def\xxt@visiblespace{
  \iffontchar\font"2423
    \expandafter\textvisiblespace
  \else
    \expandafter\xxt@visiblespace@fallback
  \fi
}
```

\xxt@visiblespace@fallback

If the current font doesn't have u2434, use Latin Modern Mono instead.

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

```
    \usefont{EU1}{lmtt}{\f@series}{\f@shape}
    \textvisiblespace
  }
}
```

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

```
\begingroup
  \catcode`\~=\active
  \gdef\xxt@vprintspaces{\catcode`\~\active\let \xxt@visiblespace}
\endgroup
```

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

```
\def\verb{
  \relax\ifmmode\hbox\else\leavevmode\null\fi
  \bgroup
    \verb@eol@error \let\do\@makeother \dospecials
    \verbatim@font\@noligs
    \@ifstar\@@sverb\@verb
}
\def\@@sverb{\xxt@vprintspaces\@sverb}
```

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

```
\AtBeginDocument{
  \fontspec_patch_verbatim:
  \fontspec_patch_moreverb:
  \fontspec_patch_fancyvrb:
  \fontspec_patch_listings:
}
```

`verbatim*` With the verbatim package.

```
\cs_set:Npn \fontspec_patch_verbatim: {
  \@ifpackageloaded{verbatim}{
    \@namedef{verbatim*}{
      \begingroup\@verbatim\xxt@vprintspaces\verbatim@start
    }
  }{
```

This is for vanilla LaTeX.

```
    \@namedef{verbatim*}{\@verbatim\xxt@vprintspaces\@sxverbatim}
  }
}
```

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

```
\cs_set:Npn \fontspec_patch_moreverb: {
  \@ifpackageloaded{moreverb}{
    \@namedef{listingcont*}{
      \def\verbatim@processline{
        \thelisting@line \global\advance\listing@line\c_one
        \the\verbatim@line\par
      }
      \@verbatim\xxt@vprintspaces\verbatim@start
    }
  }{}}
```

listings and fancvrb make things nice and easy:

```
\cs_set:Npn \fontspec_patch_fancyvrb: {
  \@ifpackageloaded{fancyvrb}{
    \let\FancyVerbSpace\xxt@visiblespace
  }{}
}
\cs_set:Npn \fontspec_patch_listings: {
  \@ifpackageloaded{listings}{
    \let\lst@visiblespace\xxt@visiblespace
  }{}
}
```

Part IX
fontspec.cfg

As an example, and to avoid upsetting people as much as possible, I'm populating the default `fontspec.cfg` file with backwards compatibility feature aliases.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% FOR BACKWARDS COMPATIBILITY WITH PREVIOUS VERSIONS %%%

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

```

Index

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

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

X

Z