The fontspec package
Font selection for XeLaTeX and LuaLaTeX

Will Robertson and Khaled Hosny
will.robertson@latex-project.org

2015/03/14 v2.4c

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.

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 document is typeset with pdfLaTeX using pre-compiled examples that have been generated by either XeTeX or LuaTeX. You may regenerate the examples by removing the `doc-files/` subdirectory and typesetting the manual with the following invocation:

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

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

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

2.2 Acknowledgements

This package could not have been possible without the early and continued support the author of XeTeX, Jonathan Kew. When I started this package, he steered me many times in the right direction.

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

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

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

3 Package loading and options

For basic use, no package options are required:

```
\usepackage{fontspec}
```

Package options will be introduced below; some preliminary details are discussed first:

xunicode Ross Moore's xunicode package is now automatically loaded for users of both XeLaTeX and LuaLaTeX. This package provides backwards compatibility with LaTeX's methods for accessing extra characters and accents (for example, `\%`, `\$`, `\textbullet`, `\"u`, and so on), plus many more Unicode characters.

XeTeX users only The xltxtra package adds some minor extra features to XeLaTeX, including, via the metalogo package, the `\XeTeX` macro to typeset the XeTeX logo. While this package was previously recommended, it serves a much smaller rôle nowadays and generally will not be required. Please consult its documentation to assess whether its features are warranted before loading it.

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 `luaotfload-tool`, which is distributed with the luaotfload package. Note that if you do not execute this script beforehand, the first time you attempt to typeset the process will pause for (up to) several minutes. (But only the first time.) Please see the luaotfload documentation for more information.

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

3.1 Maths fonts adjustments

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

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

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

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

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

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

3.2 Configuration

If you wish to customise any part of the fontspec interface, this should be done by creating your own `fontspec.cfg` file, which will be automatically loaded if it is found by XeTeX or LuaTeX. A `fontspec.cfg` file is distributed with fontspec with a small number of defaults set up within it.

To customise fontspec to your liking, use the standard `.cfg` file as a starting point or write your own from scratch, then 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 the loading of the `fontspec.cfg` file under all circumstances.

3.3 Warnings

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

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

Part I
General font selection

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

```
\fontspec{⟨font name⟩}[⟨font features⟩]
\setmainfont{⟨font name⟩}[⟨font features⟩]
\setsansfont{⟨font name⟩}[⟨font features⟩]
\setmonofont{⟨font name⟩}[⟨font features⟩]
\newfontfamily⟨cmd⟩{⟨font name⟩}[⟨font features⟩]
```

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, as shown in 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 7 on page 18, including methods for automatic scaling.

The font features argument accepts comma separated ⟨*font feature*⟩=⟨*option*⟩ lists; these are described in later:

- For general font features, see Section 7 on page 18
- For OpenType fonts, see Part II on page 22
- For XeTeX-only general font features, see Part IV on page 38
- For LuaTeX-only general font features, see Part III on page 36
- For features for AAT fonts in XeTeX, see Section 12 on page 40

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 standard names as you'd speak them out loud, such as *Times New Roman* or *Adobe Garamond*. 'Known to' in this case generally means 'exists in a standard fonts location' such as `~/Library/Fonts` on Mac OS X, or `C:\Windows\Fonts` on Windows.

The simplest example might be something like

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

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

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

4.2 By file name

XeTeX and LuaTeX also allow fonts to be loaded by file name instead of font name. When you have a very large collection of fonts, you will sometimes not wish to have them all installed in your system's font directories. In this case, it is more convenient to load them from a different location on your disk. This technique is also necessary in XeTeX when loading OpenType fonts that are present within your TeX distribution, such as `/usr/local/texlive/2013/texmf-dist/fonts/opentype/public`. Fonts in such locations are visible to XeTeX but cannot be loaded by font name, only file name; LuaTeX does not have this restriction.

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

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

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

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

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

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

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:

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

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`':

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

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 New commands to select font families

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

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

`\newfontfamily` For this reason, new commands can be created for loading a particular font family with the `\newfontfamily` command, demonstrated in Example 2. This macro should be used to create commands that would be used in the same way as `\rmfamily`, for example. If you would like to create a command that only changes the font inside its argument (i.e., the same behaviour as `\emph`) 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 being automatically selected. This is common when selecting a fancy italic font, say, that has swash features unavailable in the upright forms. `\newfontface` is used for this purpose, shown in Example 3, which is repeated in Section 12.4 on page 40.

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

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

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

5.1 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 among. The `BoldFont` and `ItalicFont` features are provided for these situations. If only one of these is used, the bold italic font is requested as the default from the *new* font. See Example 4.

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

5.1.1 Input shorthands

For those cases that the base font name is repeated, you can replace it with an asterisk. (This has been shown previously in Section 4.2 on page 6.) For example, some space can be saved instead of writing '`Baskerville SemiBold`':

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

Example 4: Explicit selection of the bold font.

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

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

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

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

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

In fact, you may specify the small caps font for each individual bold and italic shape as in

```
\fontspec{ <upright> }[
  UprightFeatures    = { SmallCapsFont={ <sc> } } ,
  BoldFeatures       = { SmallCapsFont={ <bf sc> } } ,
  ItalicFeatures     = { SmallCapsFont={ <it sc> } } ,
  BoldItalicFeatures = { SmallCapsFont={ <bf it sc> } } ,
]
Roman 123 \\ \textsc{Small caps 456}
```

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

All of the bold, italic, and small caps fonts can be loaded with different font features from the main font. See Section 6.5 for details. When an OpenType font is selected for `SmallCapsFont`, the small caps font feature is *not* automatically enabled. In this case, users should write instead, if necessary,

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

5.2 Specifically choosing the NFSS family

In LaTeX's NFSS, font families are defined with names such as 'ppl' (Palatino), 'cmr' (Computer Modern Roman), and so on, which are selected with the `\fontfamily` command:

```
\fontfamily{ppl}\selectfont
```

In fontspec, the family names are auto-generated based on the fontname of the font; for example, writing `\fontspec{Times New Roman}` for the first time would generate an internal font family name of 'TimesNewRoman(1)'.

In certain cases it is desirable to be able to choose this internal font family name so it can be re-used elsewhere for interacting with other packages that use the LaTeX's font selection interface; an example might be

```
\usepackage{fancyvrb}
\fvset{fontfamily=myverbatimfont}
```

To select a font for use in this way in fontspec use the `NFSSFamily` feature:[1]

```
\newfontfamily\verbatimfont[NFSSFamily=myverbatimfont]{Inconsolata}
```

It is then possible to write commands such as:

```
\fontfamily{myverbatimfont}\selectfont
```

which is essentially the same as writing `\verbatimfont`, or to go back to the orginal example:

```
\fvset{fontfamily=myverbatimfont}
```

Only use this feature when necessary; the in-built font switching commands that fontspec generates (such as `\verbatimfont` in the example above) are recommended in all other cases.

If you don't wish to explicitly set the NFSS family but you would like to know what it is, an alternative mechanism for package writers is introduced as part of the fontspec programming interface; see the function `\fontspec_set_family:Nnn` for details (Section 16 on page 46).

5.3 Choosing additional NFSS font faces

LaTeX's font selection scheme is more flexible than the fontspec interface discussed up until this point. It assigns to each font face a *family* (discussed above), a *series* such as bold or light or condensed, and a *shape* such as italic or slanted or small caps. The fontspec features such as `BoldFont` and so on all assign faces for the default series and shapes of the NFSS, but it's not uncommon to have font families that have multiple weights and shapes and so on.

If you set up a regular font family with the 'standard four' (upright, bold, italic, and bold italic) shapes and then want to use, say, a light font for a certain document element, many users will be perfectly happy to use `\newfontface\`⟨`switch`⟩ and use the resulting font `\`⟨`switch`⟩. In other cases, however, it is more convenient or even necessary to load additional fonts using additional NFSS specifiers.

`FontFace = {`⟨*series*⟩`}{`⟨*shape*⟩`} { Font =` ⟨*font name*⟩ , ⟨*features*⟩ `}`
`FontFace = {`⟨*series*⟩`}{`⟨*shape*⟩`}{`⟨*font name*⟩`}`

The font thus specified will inherit the font features of the main font, with optional addition ⟨*features*⟩ as requested. (Note that the optional `{`⟨*features*⟩`}` argument is still surrounded with curly braces.) Multiple `FontFace` commands may be used in a single declaration to specify multiple fonts. As an example:

[1] Thanks to Luca Fascione for the example and motivation for finally implementing this feature.

```
\setmainfont{font1.otf}[
   FontFace = {c}{n}{ font2.otf } ,
   FontFace = {c}{m}{ Font = font3.otf , Color = red }
  ]
```

Writing `\fontseries{c}\selectfont` will result in `font2` being selected, which then followed by `\fontshape{m}\selectfont` will result in `font3` being selected (in red). A font face that is defined in terms of a different series but a normal shape will attempt to find a matching small caps feature and define that face as well if appropriate. Conversely, a font faced defined in terms of a different font will not.

There are some standards for choosing shape and series codes; the LaTeX 2_ε font selection guide[2] lists series `m` for medium, `b` for bold, `bx` for bold extended, `sb` for semi-bold, and `c` for condensed. A far more comprehensive listing is included in Appendix A of Philipp Lehman's 'The Font Installation Guide'[3] covering 14 separate weights and 12 separate widths.

The `FontFace` command also interacts properly with the `SizeFeatures` command as follows: (nonsense set of font selection choices)

```
FontFace = {c}{n}{
  Font = Times ,
  SizeFeatures = {
    { Size =   -10 , Font=Georgia } ,
    { Size = 10-15}                 , % default "Font = Times"
    { Size = 15-   , Font=Cochin  } ,
  },
},
```

Note that if the first `Font` feature is omitted then each size needs its own inner `Font` declaration.

5.4 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.[4])

Note that fontspec will not change the font for general mathematics; only the upright and bold shapes will be affected. To change the font used for the mathematical symbols, see either the mathspec package or the unicode-math package.

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

[2] `texdoc fntguide`

[3] `texdoc fontinstallationguide`

[4] 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}`

However, the default text fonts may not necessarily be the ones you wish to use when typesetting maths (especially with the use of fancy ligatures and so on). For this reason, you may optionally use the commands above (in the same way as our other `\fontspec`-like commands) to explicitly state which fonts to use inside such commands as `\mathrm`. Additionally, the `\setboldmathrm` command allows 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}
```

5.5 Miscellaneous font selecting details

The optional argument — from v2.4 For the first decade of fontspec's life, optional font features were selected with a bracketed argument before the font name, as in:

```
\setmainfont[
  lots and lots ,
  and more and more ,
  an excessive number really ,
  of font features could go here
]{myfont.otf}
```

This always looked like ugly syntax to me, and the order of these arguments has now been reversed:

```
\setmainfont{myfont.otf}[
  lots and lots ,
  and more and more ,
  an excessive number really ,
  of font features could go here
]
```

I hope this doesn't cause any problems.

1. Backwards compatibility has been preserved. (In fact, you could even write

   `\fontspec[Ligatures=Rare]{myfont.otf}[Color=red]`

   if you really felt like it and both sets of features would be applied.)

2. Following standard xparse behaviour, there must be no space before the opening bracket; writing

   `\fontspec{myfont.otf}␣[Color=red]`

   will result in `[Color=red]` not being recognised an argument and therefore it will be typeset as text. When breaking over lines, write either of:

   ```
   \fontspec{myfont.otf}%
     [Color=red]
   ```

   ```
   \fontspec{myfont.otf}[
     Color=Red]
   ```

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.

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

Some default text 0123456789
Now grey, with old-style figures: 0123456789

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

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}
```

This functionality is provided with the same interface as the fixltx2e package, with a slightly different internal implementation.

6 Selecting font features

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

6.1 Default settings

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

It is sometimes useful to define font features that are applied to every subsequent font selection command. This may be defined with the `\defaultfontfeatures` command, shown in Example 5. New calls of `\defaultfontfeatures` overwrite previous ones, and defaults can be reset by calling the command with an empty argument.

`\defaultfontfeatures[`⟨*font name*⟩`]{`⟨*font features*⟩`}`

Default font features can be specified on a per-font and per-face basis by using the optional argument to `\defaultfontfeatures` as shown.[5]

```
\defaultfontfeatures[TeX Gyre Adventor]{Color=blue}
\setmainfont{TeX Gyre Adventor}% will be blue
```

[5] Internally, ⟨*font name*⟩ has all spaces removed and is converted to lowercase.

Multiple fonts may be affected by using a comma separated list of font names.

`\defaultfontfeatures[⟨\font-switch⟩]{⟨font features⟩}`

New in v2.4. Defaults can also be applied to symbolic families such as those created with the `\newfontfamily` command and for `\rmfamily`, `\sffamily`, and `\ttfamily`:

```
\defaultfontfeatures[\rmfamily,\sffamily]{Ligatures=TeX}
\setmainfont{TeX Gyre Adventor}% will use standard TeX ligatures
```

The line above to set TEX-like ligatures is now activated by *default* in `fontspec.cfg`. To reset default font features, simply call the command with an empty argument:

```
\defaultfontfeatures[\rmfamily,\sffamily]{}
\setmainfont{TeX Gyre Adventor}% will no longer use standard TeX ligatures
```

`\defaultfontfeatures+{⟨font features⟩}`
`\defaultfontfeatures+[⟨font name⟩]{⟨font features⟩}`

New in v2.4. Using the + form of the command appends the ⟨*font features*⟩ to any already-selected defaults.

6.2 Default settings from a file

In addition to the defaults that may be specified in the document as described above, when a font is first loaded, a configuration file is searched for with the name '⟨*fontname*⟩`.fontspec`'.[6]

The contents of this file can be used to specify default font features without having to have this information present within each document. ⟨*fontname*⟩ is stripped of spaces and file extensions are omitted; for example, the line above for TEX Gyre Adventor could be placed in a file called `TeXGyreAdventor.fontspec`, or for specifying options for `texgyreadventor-regular.otf` (when loading by filename), the configuration file would be `texgyreadventor-regular.fontspec`. (N.B. the lettercase of the names should match.)

This mechanism can be used to define custom names or aliases for your font collections. If you create a file `MyCharis.fontspec` containing, say,

```
\defaultfontfeatures[My Charis]
 {
  Extension = .ttf ,
  UprightFont = CharisSILR,
  BoldFont = CharisSILB,
  ItalicFont = CharisSILI,
  BoldItalicFont = CharisSILBI,
  % <any other desired options>
 }
```

you can load that custom family with `\fontspec{My Charis}` and similar. The optional argument to `\defaultfontfeatures` must match that requested by the font loading command (`\fontspec`, etc.), else the options won't take effect.

Finally, note that options for font faces can also be defined in this way. To continue the example above, here we colour the different faces:

```
\defaultfontfeatures[CharisSILR]{Color=blue}
\defaultfontfeatures[CharisSILB]{Color=red}
```

And such configuration lines can be stored either inline inside `My Charis.fontspec` or within their own `.fontspec` files; in this way, fontspec is designed to handle 'nested' configuration options as well.

[6] Located in the current folder or within a standard `texmf` location.

Example 6: A demonstration of the `\addfontfeatures` command. Note the caveat listed in the text regarding such usage.

`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{TeX Gyre Adventor}%
        [Numbers={Proportional,OldStyle}]
`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}}
```

6.3 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. Note however that the behaviour in this regard will be unreliable (subject to the font itself) if you attempt to *change* an already selected feature. *E.g.*, this sort of thing can cause troubles:

```
\addfontfeature{Numbers=OldStyle}...
\addfontfeature{Numbers=Lining}...
123
```

With both features active, how will the font render '123'? Depends on the font. In the distant future this functionality will be re-written to avoid this issue (giving '`Numbers=OldStyle`' the smarts to know to explicitly de-activate any previous instances of '`Numbers=Lining`', and vice-versa, but as I hope you can imagine this requires a fair degree of elbow grease which I haven't had available for some time now.

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

6.4 Priority of feature selection

Features defined with `\addfontfeatures` override features specified by `\fontspec`, which in turn override features specified by `\defaultfontfeatures`. If in doubt, whenever a new font is chosen for the first time, an entry is made in the transcript (`.log`) file displaying the font name and the features requested.

Example 7: Features for, say, just italics.

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 \\
```

Example 8: 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{TeX Gyre Termes}[
    UprightFeatures={Color = 220022,
         SmallCapsFeatures = {Color=115511}},
     ItalicFeatures={Color = 2244FF,
         SmallCapsFeatures = {Color=112299}},
       BoldFeatures={Color = FF4422,
         SmallCapsFeatures = {Color=992211}},
 BoldItalicFeatures={Color = 888844,
         SmallCapsFeatures = {Color=444422}},
         ]
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}
```

6.5 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.

Note that because most fonts include their small caps glyphs within the main font, features specified with `SmallCapsFeatures` are applied *in addition* to any other shape-specific features as defined above, and hence `SmallCapsFeatures` can be nested within `ItalicFeatures` and friends. Every combination of upright, italic, bold and small caps can thus be assigned individual features, as shown in the somewhat ludicrous Example 8.

Example 9: An example of specifying different font features for different sizes of font with `SizeFeatures`.

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

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

Small
Normal size
Large

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

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

6.6 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 clarify these details is shown in Example 9. A less trivial example is shown in the context of optical font sizes in Section 7.6 on page 20.

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 at all!

Example 10: Selecting colour with transparency. N.B. due to a conflict betweeen fontspec and the preview package, this example currently does not show any transparency!

```
\fontsize{48}{48}
\fontspec{TeX Gyre Bonum Bold}
{\addfontfeature{Color=FF000099}W}\kern-0.5ex
{\addfontfeature{Color=0000FF99}S}\kern-0.4ex
{\addfontfeature{Color=DDBB2299}P}\kern-0.4ex
{\addfontfeature{Color=00BB3399}R}
```

Interaction with other features For `SizeFeatures` to work with `ItalicFeatures`, `BoldFeatures`, etc., and `SmallCapsFeatures`, a strict heirarchy is required:

```
UprightFeatures =
 {
  SizeFeatures =
   {
    {
     Size = -10,
     Font = ..., % if necessary
     SmallCapsFeatures = {...},
     ... % other features for this size range
    },
    ... % other size ranges
   }
 }
```

Suggestions on simplifying this interface welcome.

7 Font independent options

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

7.1 Colour

`Color` (or `Colour`), also shown in Section 6.1 on page 13 and elsewhere, uses font specifications to set the colour of the text. The colour is defined as a triplet of two-digit Hex RGB values, with optionally another value for the transparency (where `00` is completely transparent and `FF` is opaque.) Transparency is supported by LuaLaTeX; XeLaTeX with the `xdvipdfmx` driver does not support this feature.

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

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

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

You may specify the transparency with a named colour using the `Opacity` feature which takes an decimal from zero to one corresponding to transparent to opaque respectively:

Example 11: 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
```

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

It is still possible to specify a colour in six-char hexadecimal form while defining opacity in this way, if you like.

7.2 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 Example 1. 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, the `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 11.

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.

Note that when `Scale=MatchLowercase` is used with `\setmainfont`, the new 'main' font of the document will be scaled to match the old default. This may be undesirable in some cases, so to achieve 'natural' scaling for the main font but automatically scale all other fonts selected, you may write

```
\defaultfontfeatures{ Scale = MatchLowercase }
\defaultfontfeatures[\rmfamily]{ Scale = 1}
```

One or both of these lines may be placed into a local `fontspec.cfg` file (see Section 3.2 on page 5) for this behaviour to be effected in your own documents automatically. (Also see Section 6.1 on page 13 for more information on setting font defaults.)

7.3 Interword space

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

The space in between words in a paragraph will be chosen automatically, and generally will not need to be adjusted. For those times when the precise details are important, the `WordSpace` feature is provided, which takes either a single scaling factor to scale the default value, or a triplet of comma-separated values to scale the nominal value, the stretch, and the shrink of the interword space by, respectively. (`WordSpace={x}` is the same as `WordSpace={x,x,x}`.)

Example 12: Scaling the default interword space. An exaggerated value has been chosen to emphasise the effects here.

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.

```
\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.
```

Example 13: 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{TeX Gyre Schola}[PunctuationSpace=2]
 Letters, Words. Sentences.          \par
\fontspec{TeX Gyre Schola}[PunctuationSpace=0]
 Letters, Words. Sentences.
```

7.4 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 13. Note that `PunctuationSpace=0` is *not* equivalent to `\frenchspacing`, although the difference will only be apparent when a line of text is under-full.

7.5 The hyphenation character

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

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

7.6 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.

Example 14: Explicitly choosing the hyphenation character.

EXAMPLE
HYPHENATION

EXAMPLE
HYPHEN+
ATION

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

\fontspec{Linux Libertine O}
\addfontfeature{HyphenChar=None}
\text
\addfontfeature{HyphenChar={+}}
\text
```

Example 15: A demonstration of automatic optical size selection.

Automatic optical size
Automatic optical size

```
\fontspec{Latin Modern Roman}
 Automatic optical size                  \\
\scalebox{0.4}{\Huge
 Automatic optical size}
```

OpenType fonts with optical scaling will exist in several discrete sizes, and these will be selected by XeTeX and LuaTeX *automatically* determined by the current font size as in Example 15, in which we've scaled down some large text in order to be able to compare the difference for equivalent font sizes.

The `OpticalSize` option may be used to specify a different optical size. With `OpticalSize` set to zero, no optical size font substitution is performed, as shown in Example 16.

The `SizeFeatures` feature (Section 6.6 on page 17) can be used to specify exactly which optical sizes will be used for ranges of font size. For example, something like:

```
\fontspec{Latin Modern Roman}[
  UprightFeatures = { SizeFeatures = {
    {Size=-10,     OpticalSize=8 },
    {Size= 10-14,  OpticalSize=10},
    {Size= 14-18,  OpticalSize=14},
    {Size=    18-, OpticalSize=18}}}
       ]
```

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

Latin Modern optical sizes
Latin Modern optical sizes
Latin Modern optical sizes
Latin Modern optical sizes

```
\fontspec{Latin Modern Roman 5 Regular}[OpticalSize=0]
 Latin Modern optical sizes               \\
\fontspec{Latin Modern Roman 8 Regular}[OpticalSize=0]
 Latin Modern optical sizes               \\
\fontspec{Latin Modern Roman 12 Regular}[OpticalSize=0]
 Latin Modern optical sizes               \\
\fontspec{Latin Modern Roman 17 Regular}[OpticalSize=0]
 Latin Modern optical sizes
```

Part II
OpenType

8 Introduction

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

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

A very large group of OpenType features is designed to support high quality typography in Latin, Greek, Cyrillic and other standard scripts. Examples of some font features have already been shown in previous sections; the complete set of OpenType font features supported by fontspec is described below in Section 9.

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

8.1 How to select font features

Font features are selected by a series of ⟨*feature*⟩=⟨*option*⟩ selections. Features are (usually) grouped logically; for example, all font features relating to ligatures are accessed by writing `Ligatures={...}` with the appropriate argument(s), which could be `TeX`, `Rare`, etc., as shown below in Section 9.1.

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

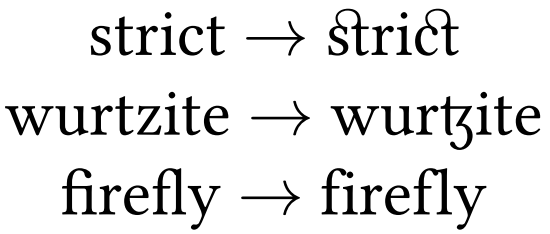
If a feature or an option is requested that the font does not have, a warning is given in the console output. As mentioned in Section 3.3 on page 5 these warnings can be suppressed by selecting the `[quiet]` package option.

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

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

∗ This feature is activated by default.

Example 17: An example of the `Ligatures` feature.




```
\def\test#1#2{%
  #2 $\to$ {\addfontfeature{#1} #2}\\}
\fontspec{Linux Libertine O}
\test{Ligatures=Historic}{strict}
\test{Ligatures=Rare}{wurtzite}
\test{Ligatures=NoCommon}{firefly}
```

9 Complete listing of OpenType font features

9.1 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. A demonstration with the Linux Libertine fonts[7] is shown in Example 17.

Note the additional features accessed with `Ligatures=TeX`. These are not actually real OpenType features, but additions provided by luaotfload (i.e., LuaTEX only) to emulate TEX's behaviour for ASCII input of curly quotes and punctuation. In XETEX this is achieved with the `Mapping` feature (see Section 11.1 on page 38) but for consistency `Ligatures=TeX` will perform the same function as `Mapping=tex-text`.

9.2 Letters

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

Petite caps are smaller than small caps. `SmallCaps` and `PetiteCaps` turn lowercase letters into the smaller caps letters, whereas the `Uppercase...` options turn the *capital* letters into

[7] http://www.linuxlibertine.org/

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

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

Example 18: Small caps from lowercase or uppercase letters.

THIS SENTENCE NO VERB
THIS SENTENCE no verb

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

the smaller caps (good, *e.g.*, for applying to already uppercase acronyms like 'NASA'). This difference is shown in Example 18. 'Unicase' is a weird hybrid of upper and lower case letters.

Note that the `Uppercase` option will (probably) not actually map letters to uppercase.[8] It is designed to select various uppercase forms for glyphs such as accents and dashes, such as shown in Example 19; note the raised position of the hyphen to better match the surrounding letters.

The `Kerning` feature also contains an `Uppercase` option, which adds a small amount of spacing in between letters (see Section 9.12 on page 31).

9.3 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 6.3 on page 15. The `Monospaced` option is useful for tabular material when digits need to be vertically aligned.

The `SlashedZero` option replaces the default zero with a slashed version to prevent confusion with an uppercase 'O', shown in Example 20.

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

Example 19: An example of the `Uppercase` option of the `Letters` feature.

UPPER-CASE example
UPPER-CASE example

```
\fontspec{Linux Libertine O}
 UPPER-CASE example \\
\addfontfeature{Letters=Uppercase}
 UPPER-CASE example
```

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

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

Example 20: The effect of the SlashedZero option.

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

0123456789 0123456789

The Arabic option (with tag anum) maps regular numerals to their Arabic script or Persian equivalents based on the current Language setting (see Section 9.18 on page 34), shown in Example 21 using the Persian Modern font, which is included in TEX Live and MiKTEX. This option is based on a LuaTEX feature of the luaotfload package, not an OpenType feature. (Thus, this feature is unavailable in XETEX.)

9.4 Contextuals

This feature refers to substitutions of glyphs that vary 'contextually' by their relative position in a word or string of characters; features such as contextual swashes are accessed via the options shown in Table 5.

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 in this feature.

9.5 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

Example 21: An example of number remapping to Arabic or Persian. (LuaTEX only.)

```
\fontspec{persian-modern-regular.ttf}%
  [Script=Arabic,Numbers=Arabic]
{\addfontfeature{Language=Arabic}
 0123456789} \\
{\addfontfeature{Language=Parsi}
 0123456789}
```

٠١٢٣٤٥٦٧٨٩
٠١٢٣۴۵۶٧٨٩

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

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

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

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

the `Inferior` feature. These are shown in Example 22

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

The realscripts package (which is also loaded by xltxtra for XeLaTeX) redefines the `\textsubscript` and `\textsuperscript` commands to use the above font features automatically, including for use in footnote labels. If this is the only feature of xltxtra you wish to use, consider loading realscripts on its own instead.

9.6 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 23.

Example 22: The `VerticalPosition` feature.

```
\fontspec{LibreCaslonText-Regular.otf}[VerticalPosition=Superior]
 Superior: 1234567890                                          \\
\fontspec{LibreCaslonText-Regular.otf}[VerticalPosition=Numerator]
 Numerator: 12345                                              \\
\fontspec{LibreCaslonText-Regular.otf}[VerticalPosition=Denominator]
 Denominator: 12345                                            \\
\fontspec{LibreCaslonText-Regular.otf}[VerticalPosition=ScientificInferior]
 Scientific Inferior: 12345
```

Superior: 1234567890
Numerator: 12345
Denominator: 12345
Scientific Inferior: 12345

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

Feature	Option	Tag
Fractions =	On	`frac`
	Alternate	`afrc`

Example 23: 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 \\
```

9.7 Stylistic Set variations

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

Two demonstrations from the Junicode font[9] are shown in Example 24 and Example 25; thanks to Adam Buchbinder for the suggestion.

Multiple stylistic sets may be selected simultaneously by writing, e.g., `StylisticSet={1,2,3}`.

The `StylisticSet` feature is a synonym of the `Variant` feature for AAT fonts. See Section 13 on page 44 for a way to assign names to stylistic sets, which should be done on a per-font basis.

9.8 Character Variants

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

For each character that can be varied, it is possible to select among possible options for that particular glyph. For example, in Example 26 a variety of glyphs for the character ‘v’ are selected, in which `5` corresponds to the character ‘v’ for this font feature, and the trailing $:\langle n \rangle$ corresponds to which variety to choose. Georg Duffner’s open source Garamond revival

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

Example 24: Insular letterforms, as used in medieval Northern Europe, for the Junicode font accessed with the `StylisticSet` feature.

Insular forms.
Inſulaꝑ ꝓoꝛmſ.

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

Example 25: 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. \\
```

Example 26: The `CharacterVariant` feature showing off Georg Duffner's open source Garamond revival font.

very
very
very
very
very
very

```
\fontspec{EB Garamond 12 Italic}                      very \\
\fontspec{EB Garamond 12 Italic}[CharacterVariant=5]  very \\
\fontspec{EB Garamond 12 Italic}[CharacterVariant=5:0] very \\
\fontspec{EB Garamond 12 Italic}[CharacterVariant=5:1] very \\
\fontspec{EB Garamond 12 Italic}[CharacterVariant=5:2] very \\
\fontspec{EB Garamond 12 Italic}[CharacterVariant=5:3] very
```

font[10] is used in this example. Character variants are specifically designed not to conflict with each other, so you can enable them individually per character as shown in Example 27. (Unlike stylistic alternates, say.)

Note that the indexing starts from zero.

9.9 Alternates

The `Alternate` feature (for the raw OpenType feature `salt`) is used to access alternate font glyphs when variations exist in the font, such as in Example 28. It uses a numerical selection,

[10] http://www.georgduffner.at/ebgaramond/

Example 27: The `CharacterVariant` feature selecting multiple variants simultaneously.

& violet
& violet
& violet
& violet

```
\fontspec{EB Garamond 12 Italic}                        \& violet \\
\fontspec{EB Garamond 12 Italic}[CharacterVariant={4}]  \& violet \\
\fontspec{EB Garamond 12 Italic}[CharacterVariant={5:2}] \& violet \\
\fontspec{EB Garamond 12 Italic}[CharacterVariant={4,5:2}] \& violet
```

Example 28: The `Alternate` feature.

A & h
A & h

```
\fontspec{Linux Libertine O}
\textsc{a} \& h \\
\addfontfeature{Alternate=0}
\textsc{a} \& h
```

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

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

starting from zero, that will be different for each font. Note that the `Style=Alternate` option is equivalent to `Alternate=0` to access the default case.

Note that the indexing starts from zero.

See Section 13 on page 44 for a way to assign names to alternates, which must be done on a per-font basis.

9.10 Style

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

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

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

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

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

M Q W
M Q W

```
\fontspec{Quattrocento Roman}
 M Q W                       \\
\addfontfeature{Style=Alternate}
 M Q W
```

Example 30: 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 31: 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 32: 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 33: 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 }
```

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

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

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

∗ This feature is activated by default.

9.11 Diacritics

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

9.12 Kerning

Specifies how inter-glyph spacing should behave. Well-made fonts include information for how differing amounts of space should be inserted between separate character pairs. This kerning space is inserted automatically but in rare circumstances you may wish to turn it off.

As briefly mentioned previously at the end of Section 9.2 on page 23, the `Uppercase` option will add a small amount of tracking between uppercase letters, seen in Example 34, which uses the Romande fonts[11] (thanks to Clea F. Rees for the suggestion). The `Uppercase` option acts separately to the regular kerning controlled by the `On`/`Off` options.

[11] http://arkandis.tuxfamily.org/adffonts.html

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

UPPERCASE EXAMPLE
UPPERCASE EXAMPLE

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

Example 35: Articifial font transformations.

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

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

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

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

Example 36: Annotation forms for OpenType fonts.

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

9.13 Font transformations

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

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

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

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

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

The `FakeBold` and `AutoFakeBold` features are only available with the XeTeX engine and will be ignored in LuaTeX.

9.14 Annotation

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

Note that the indexing starts from zero.

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

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

Example 37: Different standards for CJK ideograph presentation.

唖噛躯 妍并訝
唖噛躯 妍幷訝
啞噛軀 妍并訝

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

9.15 CJK shape

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

9.16 Character width

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

Japanese alphabetic glyphs (in Hiragana or Katakana) may be typeset proportionally, to better fit horizontal measures, or monospaced, to fit into the rigid grid imposed by

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

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

Example 38: Proportional or fixed width forms.

ようこそ　ワカヨタレソ　abcdef
ようこそ　ワカヨタレソ　ａｂｃｄｅｆ
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 39: Numbers can be compressed significantly.

—１２３２１—
-1234554321-
-123456787654321-
-1234567890987654321-

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

ideographic typesetting. In this latter case, there are also half-width forms for squeezing more kana glyphs (which are less complex than the kanji they are amongst) into a given block of space. The same features are given to roman letters in Japanese fonts, for typesetting foreign words in the same style as the surrounding text.

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

9.17 Vertical typesetting

TODO!

9.18 OpenType scripts and languages

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

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

The `Script` and `Language` features are used to designate this information. The possible options are tabulated in Table 13 on page 36 and Table 14 on page 37, respectively. When a script or language is requested that is not supported by the current font, a warning is printed in the console output.

Example 40: An example of various Scripts and Languages.

العربي	الـعـربـي
हिन्दी	हिन्दी
ലേഖ	ലഖ
મર્યાદા-સૂચક નિવેદન	મર્યાદા-સૂચક નિવેદન
നമ്മുടെ പാരമ്പര്യം	നമ്മുടെ പാരമ്പര്യം
ਆਦਿ ਸਚੁ ਜੁਗਾਦਿ ਸਚੁ	ਆਦਿ ਸਚੁ ਜੁਗਾਦਿ ਸਚੁ
தமிழ் தேடி	தமிழ் தேடி
הָרְתָה	הָרְתָה
cấp số mỗi	cấp số mỗi

```
\testfeature{Script=Arabic}{\arabictext}
\testfeature{Script=Devanagari}{\devanagaritext}
\testfeature{Script=Bengali}{\bengalitext}
\testfeature{Script=Gujarati}{\gujaratitext}
\testfeature{Script=Malayalam}{\malayalamtext}
\testfeature{Script=Gurmukhi}{\gurmukhitext}
\testfeature{Script=Tamil}{\tamiltext}
\testfeature{Script=Hebrew}{\hebrewtext}
\def\examplefont{Doulos SIL}
\testfeature{Language=Vietnamese}{\vietnamesetext}
```

Because these font features can change which features are able to be selected for the font, they are automatically selected by fontspec before all others and, if XeTeX is being used, will specifically select the OpenType renderer for this font, as described in Section 11.3 on page 39.

9.18.1 Script and Language examples

In the examples shown in Example 40, the Code2000 font[12] is used to typeset various input texts with and without the OpenType Script applied for various alphabets. The text is only rendered correctly in the second case; many examples of incorrect diacritic spacing as well as a lack of contextual ligatures and rearrangement can be seen. Thanks to Jonathan Kew, Yves Codet and Gildas Hamel for their contributions towards these examples.

9.18.2 Defining new scripts and languages

`\newfontscript`
`\newfontlanguage`

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

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

The first argument is the fontspec name, the second the OpenType tag. The advantage to using these commands rather than `\newfontfeature` (see Section 13 on page 44) is the error-checking that is performed when the script or language is requested.

[12] http://www.code2000.net/

Part III
LuaTEX-only font features

10 OpenType font feature files

An OpenType font feature file is a plain text file describing OpenType layout feature of a font in a human-readable format. The syntax of OpenType feature files is defined by Adobe[13].

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

Adding a new OpenType feature is as creating a plain text file defining the new feature and then loading it by passing its name or path to `FeatureFile`, then OpenType features defined in the file can be activated as usual.

For example, when adding one of the default features like `kern` or `liga`, no special activation is needed. On the other hand, an optional feature like `onum` or `smcp` will be activated when old style numbers or small capitals are activated, respectively. However, OpenType feature in the feature file can have any and that can be used to selectively activate the feature; for example defining a ligature feature called `mlig` and then activating it using `RawFeature` option without activating other ligatures in the font.

Figure 1 shows an example feature file. The first two lines set the script and language under which the defined features will be available, which the default language in both default and Latin scripts, respectively.

Then it defines a `liga` feature, which is a glyph substitution feature. The names starting with backslash are glyph names that is to be substituted and while the leading backslash is optional, it is used to escape glyph names when they interfere with preserved keywords. It should also be noted that glyph names are font specific and the same glyph can be named differently in different fonts.

Glyph positioning features like kerning can be defined in a similar way, but instead

[13] http://www.adobe.com/devnet/opentype/afdko/topic_feature_file_syntax.html

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

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

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

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

Figure 1: An example font feature file.

```
languagesystem DFLT dflt;
languagesystem latn dflt;

# Ligatures
feature liga {
  sub \f \i by \fi;
  sub \f \l by \fl;
} liga;

# Kerning
feature kern {
 pos \A \Y -200;
 pos \a \y -80;
} kern;
```

Example 41: XeTeX's `Mapping` feature.

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

of the keyword `sub(stitute)` the keyword `pos(ition)` is used instead. Figure 1 shows an example of adding kerning between `AY` and `ay`[14].

Lines starting with # are comments and will be ignored.

An OpenType feature file can have any number of features and can have a mix of substitution and positioning features, please refer to the full feature file specification for further documentation.

Part IV
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 41.

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

[14] The kerning is expressed in font design units which are fractions of em depending on the *units per em* value of the font, usually 1000 for PostScript fonts and 2048 for TrueType fonts.

Example 42: The LetterSpace feature.

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

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

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 42.

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

This functionality *should not be used for lowercase text*, which is spacing correctly to begin with, but it can be very useful, in small amounts, when setting small caps or all caps titles. Also see the OpenType `Uppercase` option of the `Letters` feature (Section 9.2 on page 23).

11.3 Different font technologies: AAT and OpenType

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, `OpenType`, is an open source OpenType interpreter.[15] It provides 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 `OpenType` 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, `OpenType` provides for the `Script` and `Language` features, which allow different font behaviour for different alphabets and languages; see Section 9.18 on page 34 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* `OpenType` *renderer.*

11.4 Optical font sizes

Multiple Master fonts are parameterised over orthogonal font axes, allowing continuous selection along such features as weight, width, and optical size (see **??** on page ?? 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{Minion MM Roman}[OpticalSize=11]
 MM optical size test                    \\
```

[15] v2.4: This was called '`ICU`' in previous versions of XeTeX and fontspec. Backwards compatibility is preserved.

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

12 Mac OS X's AAT fonts

> ***Warning!*** *XeTeX's implementation on Mac OS X is currently in a state of flux and the information contained below may well be wrong from 2013 onwards. There is a good chance that the features described in this section will not be available any more as XeTeX's completes its transition to a cross-platform–only application.*

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'.

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 if you do not want a ligature to be performed (the usual examples for when this might be desired are words like 'shelffull').

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 AAT fonts, they may be a combination of `Lining` or `OldStyle` and `Proportional` or `Monospaced` (the latter is good for tabular material). The synonyms `Uppercase` and `Lowercase` are equivalent to `Lining` and `OldStyle`, respectively. The differences have been shown previously in Section 6.3 on page 15.

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 43), `LineInitial`, `LineFinal`, and `Inner` (Example 44, also called 'non-final' sometimes). As non-exclusive selectors, like the ligatures, you can turn them off by prefixing their name with `No`.

Example 43: Contextual glyph for the beginnings and ends of words.

[Contextuals=WordInitial,WordFinal] *where is all the vegemite*

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

Example 44: 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{Hoefler Text}[Contextuals=Inner]
`Inner' swashes can \emph{sometimes}    \\
 contain the archaic long~s.
```

12.5 Vertical position

The `VerticalPosition` feature is used to access things like subscript (`Inferior`) and superscript (`Superior`) numbers and letters (and a small amount of punctuation, sometimes). The `Ordinal` option is (supposed to be) contextually sensitive to only raise characters that appear directly after a number. These are shown in Example 45.

The realscripts package (also loaded by xltxtra) redefines the `\textsubscript` and `\textsuperscript` commands to use the above font features, including for use in footnote labels.

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 46.

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

Some (Asian fonts predominantly) also provide for the `Alternate` feature shown in Example 47.

Example 45: 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{Skia}[VerticalPosition=Superior]
 Superior
\fontspec{Skia}[VerticalPosition=Inferior]
 Inferior                \\
\fontspec{Skia}[VerticalPosition=Ordinal]
 1st 2nd 3rd 4th 0th 8abcde
```

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

½ 5⁄6
1/2 5/6
13579⁄24680
13579/24680

```
\fontspec[Fractions=On]{Skia}
 1{^^^^2044}2 \quad 5{^^^^2044}6 \\ % fraction slash
 1/2 \quad 5/6    % regular  slash

\fontspec[Fractions=Diagonal]{Skia}
       13579{^^^^2044}24680 \\ % fraction slash
 \quad 13579/24680    % regular  slash
```

Example 47: Alternate design of pre-composed fractions.

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

```
\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
```

12.7 Variants

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

See Section 13 on page 44 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 49. See Section 13 on page 44 for a way to assign names to alternates, which should be done on a per-font basis.

Example 48: Nine variants of Zapfino.

```
\newcounter{var}
\whiledo{\value{var}<9}{%
  \edef\1{%
  \noexpand\fontspec[Variant=\thevar,
    Color=0099\thevar\thevar]{Zapfino}}\1%
  \makebox[0.75\width]{d}%
  \stepcounter{var}}
\hspace*{2cm}
```

Example 49: Alternate shape selection must be numerical.

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

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

Example 50: Vertical typesetting.

共産主義者は

共産主義者

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

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

12.9 Style

The options of the `Style` feature are defined in AAT as one of the following: `Display`, `Engraved`, `IlluminatedCaps`, `Italic`, `Ruby`,[16] `TallCaps`, or `TitlingCaps`.

Typical examples for these features are shown in Section 9.10.

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 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.

12.11 Character width

See Section 9.16 on page 33 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 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 50.

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

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

Example 51: Assigning new AAT features.

This is XeTeX by Jonathan Kew.

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

12.13 Diacritics

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

Some older fonts distributed with Mac OS X included '`O/`' *etc.* as shorthand for writing 'Ø' under the label of the `Diacritics` feature. If you come across such fonts, you'll want to turn this feature off (imagine typing `hello/goodbye` and getting 'helløgoodbye' instead!) by decomposing the two characters in the diacritic into the ones you actually want. I recommend using the proper LaTeX input conventions for obtaining such characters instead.

12.14 Annotation

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

Part V
Programming interface

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

13 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 51.

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

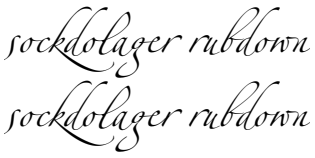
`\newopentypefeature{`⟨*feature*⟩`}{`⟨*option*⟩`}{`⟨*feature tag*⟩`}`

The synonym `\newICUfeature` is deprecated.

Here's what it would look like in practise:

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

Example 52: Assigning new arbitary features.



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

Example 53: Using raw font features directly.

PAGELLA SMALL CAPS	`\fontspec{texgyrepagella-regular.otf}[RawFeature=+smcp]` `Pagella small caps`

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

`\newfontfeature{`⟨*name*⟩`}{`⟨*input string*⟩`}`

For example, Zapfino contains the feature 'Avoid d-collisions'. To access it with this package, you could do some like that shown in Example 52. (For some reason this feature doesn't appear to be working although fontspec is doing the right thing. To be investigated.)

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

14 Going behind fontspec's back

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

Multiple features can either be included in a single declaration:

`[RawFeature=+smcp;+onum]`

or with multiple declarations:

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

15 Renaming existing features & options

`\aliasfontfeature` If you don't like the name of a particular font feature, it may be aliased to another with the `\aliasfontfeature{`⟨*existing name*⟩`}{`⟨*new name*⟩`}` command, such as shown in Example 54.

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

`\aliasfontfeatureoption` If you wish to change the name of a font feature option, it can be aliased to another with the command `\aliasfontfeatureoption{`⟨*font feature*⟩`}{`⟨*existing name*⟩`}{`⟨*new name*⟩`}`, such as shown in Example 55.

Example 54: Renaming font features.

Roman Letters *And Swash*

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

Example 55: Renaming font feature options.

S$_{\text{cientific}}$ I$_{\text{nferior}}$: $_{12345}$

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

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

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

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

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

16 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 `\l_fontspec_family_tl` macro. Otherwise, LaTeX's own `\f@family` macro can be useful here, too. The raw TeX font that is defined is stored temporarily in `\l_fontspec_font`.

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

`\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} {Tr`
Returns false if the font is not loaded by fontspec or is not an OpenType font.

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

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

`\fontspec_if_language:nnTF` Test whether the currently selected font contains the raw OpenType language tag `#2` in script `#1`. E.g.: `\fontspec_if_language:nnTF {cyrl} {SRB} {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` : LaTeX family
`#2` : fontspec features
`#3` : font name

Defines a new NFSS family from given ⟨*features*⟩ and ⟨*font*⟩, and stores the family name in the variable ⟨*family*⟩. This font family can then be selected with standard LaTeX commands `\fontfamily{`⟨*family*⟩`}\selectfont`. See the standard fontspec user commands for applications of this function.

`\fontspec_set_fontface:NNnn`
`#1` : primitive font
`#2` : LaTeX family
`#3` : fontspec features
`#4` : font name

Variant of the above in which the primitive TeX font command is stored in the variable ⟨*primitive font*⟩. If a family is loaded (with bold and italic shapes) the primitive font command will only select the regular face. This feature is designed for LaTeX programmers who need to perform subsequent font-related tests on the ⟨*primitive font*⟩.

Part VI
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.

17 Inner emphasis

fix-ltx-2e'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.

18 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 is solved by defining `\@fnsymbol` in a similar manner to the fixltx2e package.

19 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+2423: BOX OPEN, which looks like this: '␣'.

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

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

20 Discretionary hyphenation: \-

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

21 Commands for old-style and lining numbers

`\oldstylenums` `\liningnums`

LaTeX's definition of `\oldstylenums` relies on strange font encodings. We provide a fontspec-compatible alternative and while we're at it also throw in the reverse option as well. Use `\oldstylenums{`⟨*text*⟩`}` to explicitly use old-style (or lowercase) numbers in ⟨*text*⟩, and the reverse for `\liningnums{`⟨*text*⟩`}`.

Part VII
fontspec.sty and friends

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

22 'Header' code

We will eventually load the correct version of the code according to which engine we're running. As we'll see later, there are some minor differences between what we have to do in XeLaTeX and LuaLaTeX.

The expl3 module is `fontspec`.

```
⟨@@=fontspec⟩
⟨*fontspec&!xetexx&!luatex⟩
```

But for now, this is the shared code.

```
\RequirePackage{expl3}[2011/09/05]
\RequirePackage{xparse}
\ExplSyntaxOn
```

Quick fix for lualatex-math:

```
\cs_if_exist:NF \lua_now_x:n
{ \cs_set_eq:NN \lua_now_x:n \directlua }
```

Check engine and load specific modules. For LuaTeX, load only luaotfload which loads luatexbase and lualibs too.

```
\msg_new:nnn {fontspec} {cannot-use-pdftex}
{
 The~ fontspec~ package~ requires~ either~ XeTeX~ or~ LuaTeX~ to~ function.
 \\\\
 You~ must~ change~ your~ typesetting~ engine~ to,~
   e.g.,~ "xelatex"~ or~ "lualatex"\\
 instead~ of~ plain~ "latex"~ or~ "pdflatex".
}
\xetex_if_engine:F
{
 \luatex_if_engine:TF
  {
   \RequirePackage{luaotfload}[2013/05/20]
   \RequireLuaModule{fontspec}
  }
  {
   \msg_fatal:nn {fontspec} {cannot-use-pdftex}
  }
}
```

22.1 expl3 tools

22.2 Bits and pieces

Conditionals

firsttime As `\keys_set:nn` 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 (initialised true).

```
\bool_new:N \l_@@_firsttime_bool
\bool_new:N \l_@@_nobf_bool
\bool_new:N \l_@@_noit_bool
\bool_new:N \l_@@_nosc_bool

\bool_new:N \l_@@_tfm_bool
\bool_new:N \l_@@_atsui_bool
\bool_new:N \l_@@_ot_bool
\bool_new:N \l_@@_mm_bool
\bool_new:N \l_@@_graphite_bool
```

For dealing with legacy maths

```
\bool_new:N \g_@@_math_euler_bool
\bool_new:N \g_@@_math_lucida_bool
\bool_new:N \g_@@_pkg_euler_loaded_bool
```

For package options:

```
\bool_new:N \g_@@_cfg_bool
\bool_new:N \g_@@_math_bool
```

Counters

```
\int_new:N \l_fontspec_script_int
\int_new:N \l_fontspec_language_int
\int_new:N \l_fontspec_strnum_int
```

Other variables

```
\fp_new:N \l_@@_tmpa_fp
\fp_new:N \l_@@_tmpb_fp
\dim_new:N \l_@@_tmpa_dim
\dim_new:N \l_@@_tmpb_dim
\dim_new:N \l_@@_tmpc_dim

\tl_set:Nx \c_colon_str { \tl_to_str:N : }
\cs_set:Npn \use_v:nnnnn #1#2#3#4#5 {#5}
\cs_set:Npn \use_iv:nnnnn #1#2#3#4#5 {#4}
```

Need these:

```
\cs_generate_variant:Nn \str_if_eq:nnTF {nv}
\cs_generate_variant:Nn \int_set:Nn {Nv}
\cs_generate_variant:Nn \tl_gset:Nn {cV}
\cs_generate_variant:Nn \keys_set:nn {nx}
\cs_generate_variant:Nn \keys_set_known:nnN {nx}
```

`\@@_int_mult_truncate:Nn` Missing in expl3, IMO.

```
\cs_new:Nn \@@_int_mult_truncate:Nn
  {
    \int_set:Nn #1 { \__dim_eval:w #2 #1 \__dim_eval_end: }
  }
```

22.3 Error/warning/info messages

Shorthands for messages:

```
\cs_new:Npn \@@_error:n     { \msg_error:nn     {fontspec} }
\cs_new:Npn \@@_error:nx    { \msg_error:nnx    {fontspec} }
\cs_new:Npn \@@_warning:n   { \msg_warning:nn   {fontspec} }
\cs_new:Npn \@@_warning:nx  { \msg_warning:nnx  {fontspec} }
\cs_new:Npn \@@_warning:nxx { \msg_warning:nnxx {fontspec} }
\cs_new:Npn \@@_info:n      { \msg_info:nn      {fontspec} }
\cs_new:Npn \@@_info:nx     { \msg_info:nnx     {fontspec} }
\cs_new:Npn \@@_info:nxx    { \msg_info:nnxx    {fontspec} }
\cs_new:Npn \@@_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} {font-not-found}
 {
  The~ font~ "#1"~ cannot~ be~ found.
 }
 {
  A~font~might~not~be~found~for~many~reasons.\\
  Check~the~spelling,~where~the~font~is~installed~etc.~etc.\\\\
  When~in~doubt,~ask~someone~for~help!
 }
\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}
 {
  '\l_fontspec_fontname_tl'~ does~ not~ contain~ glyph~ #1.
 }
\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-latn}
 {
  Font~ '\l_fontspec_fontname_tl'~ does~ not~ contain~ script~ '#1'.\\
  'Latin'~ script~ used~ instead.
 }
\msg_new:nnn {fontspec} {script-not-exist}
 {
  Font~ '\l_fontspec_fontname_tl'~ does~ not~ contain~ script~ '#1'.
 }
\msg_new:nnn {fontspec} {aat-feature-not-exist}
 {
  '\l_keys_key_tl=\l_keys_value_tl'~ feature~ not~ supported~
  for~ AAT~ font~ '\l_fontspec_fontname_tl'.
 }
\msg_new:nnn {fontspec} {aat-feature-not-exist-in-font}
 {
  AAT~ feature~ '\l_keys_key_tl=\l_keys_value_tl'~ (#1)~ not~ available~
  in~ font~ '\l_fontspec_fontname_tl'.
 }
\msg_new:nnn {fontspec} {icu-feature-not-exist}
 {
  '\l_keys_key_tl=\l_keys_value_tl'~ feature~ not~ supported~
  for~ OpenType~ font~ '\l_fontspec_fontname_tl'
 }
\msg_new:nnn {fontspec} {icu-feature-not-exist-in-font}
 {
  OpenType~ feature~ '\l_keys_key_tl=\l_keys_value_tl'~ (#1)~ not~ available~
  for~ font~ '\l_fontspec_fontname_tl'~
```

```
 with~ script~ '\l_@@_script_name_tl'~ and~ language~ '\l_@@_lang_name_tl'.
 }
\msg_new:nnn {fontspec} {no-opticals}
 {
 '\l_fontspec_fontname_tl'~ doesn't~ appear~ to~ have~ an~ Optical~ Size~ axis.
 }
\msg_new:nnn {fontspec} {language-not-exist}
 {
  Language~ '#1'~ not~ available~
  for~ font~ '\l_fontspec_fontname_tl'~
  with~ script~ '\l_@@_script_name_tl'.\\
  'Default'~ language~ used~ instead.
 }
\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.
 }
\msg_new:nnn {fontspec} {fakebold-only-xetex}
 {
 The~ "FakeBold"~ and~ "AutoFakeBold"~ options~ are~ only~ available~ with~ XeLaTeX.\\
  Option~ ignored.
 }
```

Info messages:

```
\msg_new:nnn {fontspec} {defining-font}
 {
  Font~ family~'\l_fontspec_family_tl'~ created~ for~ font~ '#2'~
  with~ options~ [\l_@@_all_features_clist].\\
  \\
  This~ font~ family~ consists~ of~ the~ following~ shapes:
  \l_fontspec_defined_shapes_tl
 }
\msg_new:nnn {fontspec} {no-font-shape}
 {
  Could~ not~ resolve~ font~ #1~ (it~ probably~ doesn't~ exist).
```

```
 }
\msg_new:nnn {fontspec} {set-scale}
 {
  \l_fontspec_fontname_tl\space scale ~=~ \l_@@_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~ \l_fontspec_fontname_tl\space does~ not~ contain~ any~ OpenType~ ‘Script’~ information.
 }
\msg_new:nnn {fontspec} {opa-twice}
 {
  Opacity~ set~ twice,~ in~ both~ Colour~ and~ Opacity.\\
  Using~ specification~ "Opacity=#1".
 }
\msg_new:nnn {fontspec} {opa-twice-col}
 {
  Opacity~ set~ twice,~ in~ both~ Opacity~ and~ Colour.\\
  Using~ an~ opacity~ specification~ in~ hex~ of~ "#1/FF".
 }
\msg_new:nnn {fontspec} {bad-colour}
 {
  Bad~ colour~ declaration~ "#1".~
  Colour~ must~ be~ one~ of:\\
  *~ a~ named~ xcolor~ colour\\
  *~ a~ six-digit~ hex~ colour~ RRGGBB\\
  *~ an~ eight-digit~ hex~ colour~ RRGGBBTT~ with~ opacity
 }
```

22.4 Option processing

```
\DeclareOption{cm-default}
 { \@@_warning:n {cm-default-obsolete} }
\DeclareOption{math}{\bool_set_true:N \g_@@_math_bool}
\DeclareOption{no-math}{\bool_set_false:N \g_@@_math_bool}
\DeclareOption{config}{\bool_set_true:N \g_@@_cfg_bool}
\DeclareOption{no-config}{\bool_set_false:N \g_@@_cfg_bool}
\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*
```

22.5 Packages

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}
\luatex_if_engine:T { \RequirePackage{fontspec-luatex} \endinput }
\xetex_if_engine:T  { \RequirePackage{fontspec-xetex}  \endinput }
```

⟨/fontspec&!xetexx&!luatex⟩

23 The main package code

That was the driver, and now the fun starts.

⟨∗fontspec & (xetexx | luatex)⟩
```
\ExplSyntaxOn
```

23.1 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.

⟨xetexx⟩
```
\tl_set:Nn \g_fontspec_encoding_tl {EU1}
```
⟨luatex⟩
```
\tl_set:Nn \g_fontspec_encoding_tl {EU2}
\tl_set:Nn \rmdefault {lmr}
\tl_set:Nn \sfdefault {lmss}
\tl_set:Nn \ttdefault {lmtt}
\RequirePackage[\g_fontspec_encoding_tl]{fontenc}
\tl_set_eq:NN \UTFencname \g_fontspec_encoding_tl % for xunicode
```

Dealing with a couple of the problems introduced by babel:

```
\tl_set_eq:NN \cyrillicencoding \g_fontspec_encoding_tl
\tl_set_eq:NN \latinencoding    \g_fontspec_encoding_tl
\AtBeginDocument
{
 \tl_set_eq:NN \cyrillicencoding \g_fontspec_encoding_tl
 \tl_set_eq:NN \latinencoding    \g_fontspec_encoding_tl
}
```

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

xunicode Now we load xunicode, working around its internal XETEX check when under LuaTEX.

⟨xetexx⟩
```
\RequirePackage{xunicode}
```
⟨∗luatex⟩
```
\cs_set_eq:NN \fontspec_tmp: \XeTeXpicfile
\cs_set:Npn \XeTeXpicfile {}
\RequirePackage{xunicode}
\cs_set_eq:NN \XeTeXpicfile \fontspec_tmp:
```
⟨/luatex⟩

23.2 User commands

This section contains the definitions of the commands detailed in the user documentation. Only the 'top level' definitions of the commands are contained herein; they all use or define macros which are defined or used later on in Section 23.5 on page 68.

23.2.1 Helper macros for argument mangling

```
\cs_new:Nn \@@_pass_args:nnn
 {
  \IfNoValueTF {#2}
   { \@@_post_arg:w {#1} {#3} }
   { #1 {#2} {#3} }
 }
\NewDocumentCommand \@@_post_arg:w { m m O{} }
 { #1 {#3} {#2} }
```

23.2.2 Font selection

`\fontspec` This is the main command of the package that selects fonts with various features. It takes two arguments: the font name and the optional requested features of that font. Then this new font family is selected.

```
\NewDocumentCommand \fontspec { o m }
 { \@@_pass_args:nnn \@@_fontspec:nn {#1} {#2} }

\cs_new:Nn \@@_fontspec:nn
 {
  \fontencoding {\g_fontspec_encoding_tl}
  \fontspec_set_family:Nnn \f@family {#1}{#2}
  \selectfont
  \ignorespaces
 }
```

`\setmainfont` `\setsansfont` `\setmonofont` The following three macros perform equivalent operations setting the default font 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 }
 { \@@_pass_args:nnn \@@_setmainfont:nn {#1} {#2} }

\cs_new:Nn \@@_setmainfont:nn
 {
  \fontspec_set_family:Nnn \rmdefault {#1}{#2}
  \normalfont
  \ignorespaces
 }

\DeclareDocumentCommand \setsansfont { o m }
 { \@@_pass_args:nnn \@@_setsansfont:nn {#1} {#2} }

\cs_new:Nn \@@_setsansfont:nn
 {
  \fontspec_set_family:Nnn \sfdefault {#1}{#2}
  \normalfont
```

```
  \ignorespaces
 }

\DeclareDocumentCommand \setmonofont { o m }
 { \@@_pass_args:nnn \@@_setmonofont:nn {#1} {#2} }

\cs_new:Nn \@@_setmonofont:nn
 {
  \fontspec_set_family:Nnn \ttdefault {#1}{#2}
  \normalfont
  \ignorespaces
 }
```

\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 \setmainfont 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.

```
\tl_new:N \g_@@_mathrm_tl
\tl_new:N \g_@@_bfmathrm_tl
\tl_new:N \g_@@_mathsf_tl
\tl_new:N \g_@@_mathtt_tl
\DeclareDocumentCommand \setmathrm { o m }
 { \@@_pass_args:nnn \@@_setmathrm:nn {#1} {#2} }

\cs_new:Nn \@@_setmathrm:nn
 {
  \fontspec_set_family:Nnn \g_@@_mathrm_tl {#1} {#2}
 }

\DeclareDocumentCommand \setboldmathrm { o m }
 { \@@_pass_args:nnn \@@_setboldmathrm:nn {#1} {#2} }

\cs_new:Nn \@@_setboldmathrm:nn
 {
  \fontspec_set_family:Nnn \g_@@_bfmathrm_tl {#1} {#2}
 }

\DeclareDocumentCommand \setmathsf { o m }
 { \@@_pass_args:nnn \@@_setmathsf:nn {#1} {#2} }

\cs_new:Nn \@@_setmathsf:nn
 {
  \fontspec_set_family:Nnn \g_@@_mathsf_tl {#1} {#2}
 }

\DeclareDocumentCommand \setmathtt { o m }
 { \@@_pass_args:nnn \@@_setmathtt:nn {#1} {#2} }

\cs_new:Nn \@@_setmathtt:nn
 {
```

```
  \fontspec_set_family:Nnn \g_@@_mathtt_tl {#1} {#2}
 }
\@onlypreamble\setmathrm
\@onlypreamble\setboldmathrm
\@onlypreamble\setmathsf
\@onlypreamble\setmathtt
```

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

```
\tl_set:Nn \g_@@_mathrm_tl {\rmdefault}
\tl_set:Nn \g_@@_mathsf_tl {\sfdefault}
\tl_set:Nn \g_@@_mathtt_tl {\ttdefault}
```

`\newfontfamily` `\newfontface` This macro takes the arguments of `\fontspec` with a prepended ⟨*instance cmd*⟩. This command is used when a specific font instance needs to be referred to repetitively (*e.g.*, in a section heading) since continuously calling `\fontspec_select:nn` is inefficient because it must parse the option arguments every time.

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

```
\DeclareDocumentCommand \newfontfamily { m o m }
 { \@@_pass_args:nnn { \@@_newfontfamily:Nnn #1 } {#2} {#3} }

\cs_new:Nn \@@_newfontfamily:Nnn
 {
  \fontspec_set_family:cnn { g_@@_ \cs_to_str:N #1 _family } {#2} {#3}
  \use:x
   {
    \exp_not:N \DeclareRobustCommand \exp_not:N #1
     {
      \exp_not:N \fontencoding {\g_fontspec_encoding_tl}
     \exp_not:N \fontfamily { \use:c {g_@@_ \cs_to_str:N #1 _family} } \exp_not:N \selectfont
     }
   }
 }
```

`\newfontface` uses the fact that if the argument to `BoldFont`, etc., is empty (*i.e.*, `BoldFont={}`), then no bold font is searched for.

```
\DeclareDocumentCommand \newfontface { m o m }
 { \@@_pass_args:nnn { \@@_newfontface:Nnn #1 } {#2} {#3} }

\cs_new:Nn \@@_newfontface:Nnn
 {
  \newfontfamily #1 [ BoldFont={},ItalicFont={},SmallCapsFont={},#2 ] {#3}
 }
```

23.2.3 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 `\g_fontspec_default_fontopts_tl` (initialised empty), which is concatenated with the individual macro choices in the [...] macro.

```
\clist_new:N \g_@@_default_fontopts_clist
\prop_new:N \g_@@_fontopts_prop
```

```
\DeclareDocumentCommand \defaultfontfeatures { t+ o m }
{
 \IfNoValueTF {#2}
  { \@@_set_default_features:nn {#1} {#3} }
  { \@@_set_font_default_features:nnn {#1} {#2} {#3} }
 \ignorespaces
}
\cs_new:Nn \@@_set_default_features:nn
{
 \IfBooleanTF {#1} \clist_put_right:Nn \clist_set:Nn
  \g_@@_default_fontopts_clist {#2}
}
```

The optional argument specifies a font identifier. Branch for either (a) single token input such as `\rmdefault`, or (b) otherwise assume its a fontname. In that case, strip spaces and file extensions and lower-case to ensure consistency.

```
\cs_new:Nn \@@_set_font_default_features:nnn
{
 \clist_map_inline:nn {#2}
  {
   \tl_if_single:nTF {##1}
   { \tl_set:No \l_@@_tmp_tl { \cs:w g_@@_ \cs_to_str:N ##1 _family\cs_end: } }
    { \@@_sanitise_fontname:Nn \l_@@_tmp_tl {##1} }

   \IfBooleanTF {#1}
    {
     \prop_get:NVNF \g_@@_fontopts_prop \l_@@_tmp_tl \l_@@_tmpb_tl
      { \tl_clear:N \l_@@_tmpb_tl }
     \tl_put_right:Nn \l_@@_tmpb_tl {#3,}
     \prop_gput:NVV   \g_@@_fontopts_prop \l_@@_tmp_tl \l_@@_tmpb_tl
    }
    {
     \tl_if_empty:nTF {#3}
      { \prop_gremove:NV \g_@@_fontopts_prop \l_@@_tmp_tl }
      { \prop_put:NVn    \g_@@_fontopts_prop \l_@@_tmp_tl {#3,} }
    }
  }
}
```

`\@@_sanitise_fontname:Nn` Assigns font name `#2` to token list variable `#1` and strips extension(s) from it in the case of an external font. We strip spaces for luatex for consistency with luaotfload, although I'm not sure this is necessary any more. At one stage this also lowercased the name, but this step has been removed unless someone can remind me why it was necessary.

```
\cs_new:Nn \@@_sanitise_fontname:Nn
{
 \tl_set:Nx #1 {#2}
⟨luatex⟩  \tl_remove_all:Nn #1 {~}
 \clist_map_inline:Nn \l_@@_extensions_clist
  { \tl_remove_once:Nn #1 {##1} }
}
```

`\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 `\g_fontspec_default_fontopts_tl` is emptied inside the group; this is allowed as `\l_fontspec_family_tl` 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.

```
\bool_new:N \l_@@_disable_defaults_bool
\DeclareDocumentCommand \addfontfeatures {m}
{
 \fontspec_if_fontspec_font:TF
  {
   \group_begin:
     \prop_get:cnN {g_@@_ \f@family _prop} {options} \l_@@_options_tl
     \prop_get:cnN {g_@@_ \f@family _prop} {fontname} \l_@@_fontname_tl
     \bool_set_true:N \l_@@_disable_defaults_bool
     \use:x
      {
       \exp_not:N \fontspec_select:nn
         { \l_@@_options_tl , #1 } {\l_@@_fontname_tl}
      }
   \group_end:
   \fontfamily\l_fontspec_family_tl\selectfont
  }
  {
   \@@_warning:n {addfontfeatures-ignored}
  }
 \ignorespaces
}
\cs_set_eq:NN \addfontfeature \addfontfeatures
```

23.2.4 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.

```
\DeclareDocumentCommand \newfontfeature {mm}
{
 \keys_define:nn { fontspec }
  {
   #1 .code:n =
    {
     \@@_update_featstr:n {#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}
{
 \keys_if_exist:nnF { fontspec } {#1}
   { \@@_define_font_feature:n {#1} }
 \keys_if_choice_exist:nnnT {fontspec} {#1} {#2}
   { \@@_warning:nxx {feature-option-overwrite} {#1} {#2} }
 \@@_define_feature_option:nnnnn {#1}{#2}{#3}{#4}{}
}
```

`\newopentypefeature` `\newICUfeature`

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

```
\DeclareDocumentCommand \newopentypefeature {mmm}
{
 \keys_if_exist:nnF { fontspec / options } {#1}
   { \@@_define_font_feature:n {#1} }
 \keys_if_choice_exist:nnnT {fontspec} {#1} {#2}
   { \@@_warning:nxx {feature-option-overwrite} {#1} {#2} }
 \@@_define_feature_option:nnnnn {#1}{#2}{}{}{#3}
}
\cs_set_eq:NN \newICUfeature \newopentypefeature % deprecated
```

`\aliasfontfeature` `\aliasfontfeatureoption`

User commands for renaming font features and font feature options.

```
\DeclareDocumentCommand \aliasfontfeature {mm}
{
 \clist_map_inline:nn
  { fontspec, fontspec-preparse, fontspec-preparse-external,
    fontspec-preparse-nested, fontspec-renderer }
  {
    \keys_if_exist:nnT {##1} {#1}
     {
       \clist_map_break:n
         {
           \@@_alias_font_feature:nnn {##1} {#1} {#2}
           \use_none_delimit_by_q_nil:w
         }
     }
  }

 % this executes if no match was found:
 \@@_warning:nx {rename-feature-not-exist} {#1}

 % jump to here if a match:
 \use_none:n
 \q_nil
}

\cs_set:Nn \@@_alias_font_feature:nnn
 {
   \keys_define:nn {#1}
     { #3 .code:n = { \keys_set:nn {#1} { #2 = {##1} } } }
 }
```

```
\DeclareDocumentCommand \aliasfontfeatureoption {mmm}
  { \keys_define:nn { fontspec / #1 } { #3 .meta:n = {#2} } }
```

\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}
}
\keys_define:nn { fontspec } { Script .choice: }
\cs_new:Nn \fontspec_new_script:nn
{
 \keys_define:nn { fontspec } { Script / #1 .code:n =
   \fontspec_check_script:nTF {#2}
    {
     \tl_set:Nn \l_fontspec_script_tl {#2}
     \int_set:Nn \l_fontspec_script_int {\l_fontspec_strnum_int}
    }
    {
     \fontspec_check_script:nTF {latn}
      {
       \@@_warning:nx {script-not-exist-latn} {#1}
       \keys_set:nn {fontspec} {Script=Latin}
      }
      {
       \@@_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}
}
\keys_define:nn { fontspec } { Language .choice: }
\cs_new:Nn \fontspec_new_lang:nn
{
 \keys_define:nn { fontspec } { Language / #1 .code:n =
   \fontspec_check_lang:nTF {#2}
    {
     \tl_set:Nn \l_fontspec_lang_tl {#2}
```

```
      \int_set:Nn \l_fontspec_language_int {\l_fontspec_strnum_int}
     }
     {
      \@@_warning:nx {language-not-exist} {#1}
      \keys_set:nn { fontspec } { Language = Default }
     }
  }
}
```

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

```
\DeclareDocumentCommand \DeclareFontsExtensions {m}
{
 \clist_set:Nn \l_@@_extensions_clist { #1 }
 \tl_remove_all:Nn \l_@@_extensions_clist {~}
}
\DeclareFontsExtensions{.otf,.ttf,.OTF,.TTF,.ttc,.TTC,.dfont}
```

23.3 Programmer's interface

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

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

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

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

```
\prg_new_conditional:Nnn \fontspec_if_fontspec_font: {TF,T,F}
{
 \cs_if_exist:cTF {g_@@_ \f@family _prop} \prg_return_true: \prg_return_false:
}
```

`\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}
{
 \fontspec_if_fontspec_font:TF
  {
   \prop_get:cnN {g_@@_ \f@family _prop} {fontdef} \l_@@_fontdef_tl
   \@@_font_set:Nnn \l_fontspec_font {\l_@@_fontdef_tl} {\f@size pt}
   \bool_if:NTF \l_@@_atsui_bool
    {
     \fontspec_make_AAT_feature_string:nnTF {#1}{#2}
       \prg_return_true: \prg_return_false:
    }
    {
     \prg_return_false:
    }
  }
  {
   \prg_return_false:
```

```
  }
}
```

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

```
\prg_new_conditional:Nnn \fontspec_if_opentype: {TF,T,F}
{
 \fontspec_if_fontspec_font:TF
  {
   \prop_get:cnN {g_@@_ \f@family _prop} {fontdef} \l_@@_fontdef_tl
   \@@_font_set:Nnn \l_fontspec_font {\l_@@_fontdef_tl} {\f@size pt}
   \@@_set_font_type:
   \bool_if:NTF \l_@@_ot_bool \prg_return_true: \prg_return_false:
  }
  {
   \prg_return_false:
  }
}
```

`\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.

```
\prg_new_conditional:Nnn \fontspec_if_feature:n {TF,T,F}
{
 \fontspec_if_fontspec_font:TF
  {
   \prop_get:cnN {g_@@_ \f@family _prop} {fontdef} \l_@@_fontdef_tl
   \@@_font_set:Nnn \l_fontspec_font {\l_@@_fontdef_tl} {\f@size pt}
   \@@_set_font_type:
   \bool_if:NTF \l_@@_ot_bool
    {
     \prop_get:cnN {g_@@_ \f@family _prop} {script-num} \l_@@_tmp_tl
     \int_set:Nn \l_fontspec_script_int {\l_@@_tmp_tl}

     \prop_get:cnN {g_@@_ \f@family _prop} {lang-num} \l_@@_tmp_tl
     \int_set:Nn \l_fontspec_language_int {\l_@@_tmp_tl}

    \prop_get:cnN {g_@@_ \f@family _prop} {script-tag}  \l_fontspec_script_tl
     \prop_get:cnN {g_@@_ \f@family _prop} {lang-tag}  \l_fontspec_lang_tl

    \fontspec_check_ot_feat:nTF {#1} {\prg_return_true:} {\prg_return_false:}
    }
    {
     \prg_return_false:
    }
  }
  {
   \prg_return_false:
  }
}
```

`\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} {Tr`

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}
{
 \fontspec_if_fontspec_font:TF
  {
   \prop_get:cnN {g_@@_ \f@family _prop} {fontdef} \l_@@_fontdef_tl
   \@@_font_set:Nnn \l_fontspec_font {\l_@@_fontdef_tl} {\f@size pt}
   \@@_set_font_type:
   \bool_if:NTF \l_@@_ot_bool
    {
     \fontspec_iv_str_to_num:Nn \l_fontspec_script_int {#1}
     \fontspec_iv_str_to_num:Nn \l_fontspec_language_int {#2}
     \fontspec_check_ot_feat:nTF {#3} \prg_return_true: \prg_return_false:
    }
    { \prg_return_false: }
  }
  { \prg_return_false: }
}
```

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

```
\prg_new_conditional:Nnn \fontspec_if_script:n {TF,T,F}
{
 \fontspec_if_fontspec_font:TF
  {
   \prop_get:cnN {g_@@_ \f@family _prop} {fontdef} \l_@@_fontdef_tl
   \@@_font_set:Nnn \l_fontspec_font {\l_@@_fontdef_tl} {\f@size pt}
   \@@_set_font_type:
   \bool_if:NTF \l_@@_ot_bool
    {
     \fontspec_check_script:nTF {#1} \prg_return_true: \prg_return_false:
    }
    { \prg_return_false: }
  }
  { \prg_return_false: }
}
```

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

```
\prg_new_conditional:Nnn \fontspec_if_language:n {TF,T,F}
{
 \fontspec_if_fontspec_font:TF
  {
   \prop_get:cnN {g_@@_ \f@family _prop} {fontdef} \l_@@_fontdef_tl
   \@@_font_set:Nnn \l_fontspec_font {\l_@@_fontdef_tl} {\f@size pt}
   \@@_set_font_type:
   \bool_if:NTF \l_@@_ot_bool
    {
     \prop_get:cnN {g_@@_ \f@family _prop} {script-num} \l_@@_tmp_tl
```

```
      \int_set:Nn \l_fontspec_script_int {\l_@@_tmp_tl}
     \prop_get:cnN {g_@@_ \f@family _prop} {script-tag}  \l_fontspec_script_tl

      \fontspec_check_lang:nTF {#1} \prg_return_true: \prg_return_false:
     }
     { \prg_return_false: }
   }
   { \prg_return_false: }
}
```

`\fontspec_if_language:nnTF` Test whether the currently selected font contains the raw OpenType language tag #2 in script #1. E.g.: `\fontspec_if_language:nnTF {cyrl} {SRB} {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}
{
 \fontspec_if_fontspec_font:TF
  {
   \prop_get:cnN {g_@@_ \f@family _prop} {fontdef} \l_@@_fontdef_tl
   \@@_font_set:Nnn \l_fontspec_font {\l_@@_fontdef_tl} {\f@size pt}
   \@@_set_font_type:
   \bool_if:NTF \l_@@_ot_bool
    {
     \tl_set:Nn \l_fontspec_script_tl {#1}
     \fontspec_iv_str_to_num:Nn \l_fontspec_script_int {#1}
     \fontspec_check_lang:nTF {#2} \prg_return_true: \prg_return_false:
    }
    { \prg_return_false: }
  }
  { \prg_return_false: }
}
```

`fontspec_if_current_script:nTF` Test 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}
{
 \fontspec_if_fontspec_font:TF
  {
   \prop_get:cnN {g_@@_ \f@family _prop} {fontdef} \l_@@_fontdef_tl
   \@@_font_set:Nnn \l_fontspec_font {\l_@@_fontdef_tl} {\f@size pt}
   \@@_set_font_type:
   \bool_if:NTF \l_@@_ot_bool
    {
     \prop_get:cnN {g_@@_ \f@family _prop} {script-tag}  \l_@@_tmp_tl
     \str_if_eq:nVTF {#1}  \l_@@_tmp_tl
       {\prg_return_true:} {\prg_return_false:}
    }
    { \prg_return_false: }
  }
  { \prg_return_false: }
}
```

`ontspec_if_current_language:nTF` Test 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}
```

```
{
 \fontspec_if_fontspec_font:TF
  {
   \prop_get:cnN {g_@@_ \f@family _prop} {fontdef} \l_@@_fontdef_tl
   \@@_font_set:Nnn \l_fontspec_font {\l_@@_fontdef_tl} {\f@size pt}
   \@@_set_font_type:
   \bool_if:NTF \l_@@_ot_bool
    {
     \prop_get:cnN {g_@@_ \f@family _prop} {lang-tag}  \l_@@_tmp_tl
     \str_if_eq:nVTF {#1} \l_@@_tmp_tl
       {\prg_return_true:} {\prg_return_false:}
    }
    { \prg_return_false: }
  }
  { \prg_return_false: }
}
```

`\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:Nn \fontspec_set_family:Nnn
{
 \tl_set:Nn \l_@@_family_label_tl { #1 }
 \fontspec_select:nn {#2}{#3}
 \tl_set_eq:NN #1 \l_fontspec_family_tl
}
\cs_generate_variant:Nn \fontspec_set_family:Nnn {c}
```

`\fontspec_set_fontface:NNnn`

```
\cs_new:Nn \fontspec_set_fontface:NNnn
{
 \tl_set:Nn \l_@@_family_label_tl { #1 }
 \fontspec_select:nn {#3}{#4}
 \tl_set_eq:NN #1 \l_fontspec_font
 \tl_set_eq:NN #2 \l_fontspec_family_tl
}
```

23.4 expl3 interface for font loading

```
\cs_set:Nn \@@_fontwrap:n { "#1" }
```

Beginnings of an 'l3font', I guess:

```
\cs_if_free:NT \font_set_eq:NN
{
 \cs_set_eq:NN \font_set_eq:NN \tex_let:D
```

```
  \cs_set:Npn \font_set:Nnn #1#2#3
   {
    \font #1 = #2 ~at~ #3\scan_stop:
   }
  \cs_set:Npn \font_gset:Nnn #1#2#3
   {
    \global \font #1 = #2 ~at~ #3 \scan_stop:
   }
  \cs_set:Npn \font_suppress_not_found_error:
⟨xetexx⟩    {\suppressfontnotfounderror=1}
⟨luatex⟩    {\luatexsuppressfontnotfounderror=1}
  \prg_set_conditional:Nnn \@@_font_if_null:N {p,TF,T,F}
   {
    \ifx #1 \nullfont
      \prg_return_true:
    \else
      \prg_return_false:
    \fi
   }
}
```

`spec_set:Nnn,\fontspec_gset:Nnn` Wrapper around `\font_set:Nnn` and `\font_gset:Nnn`.

```
\cs_new:Nn \@@_font_set:Nnn
{
 \font_set:Nnn #1 {\@@_fontwrap:n {#2}} {#3}
}
\cs_new:Nn \@@_font_gset:Nnn
{
 \font_gset:Nnn #1 {\@@_fontwrap:n {#2}} {#3}
}
```

`\font_glyph_if_exist:NnTF`

```
\prg_new_conditional:Nnn \font_glyph_if_exist:Nn {p,TF,T,F}
{
 \etex_iffontchar:D #1 #2 \scan_stop:
   \prg_return_true:
 \else:
   \prg_return_false:
 \fi:
}
```

23.5 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 `\l_fontspec_family_tl`. The TEX '`\font`' command is (globally) stored in `\l_fontspec_font`.

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 cleared.

Some often-used variables to know about:

- `\l_fontspec_fontname_tl` is used as the generic name of the font being defined.
- `\l_@@_fontid_tl` is the unique identifier of the font with all its features.
- `\l_fontspec_fontname_up_tl` is the font specifically to be used as the upright font.
- `\l_@@_basename_tl` is the (immutable) original argument used for *-replacing.
- `\l_fontspec_font` is the plain TEX font of the upright font requested.

```
\cs_set:Nn \fontspec_select:nn
{
 \group_begin:
 \font_suppress_not_found_error:
 \@@_init:

 \tl_set:Nx \l_fontspec_fontname_tl    {#2}
 \tl_set:Nx \l_fontspec_fontname_up_tl {#2}
 \tl_set:Nx \l_@@_basename_tl          {#2}

 \@@_load_external_fontoptions:Nn \l_fontspec_fontname_tl {#2}
 \@@_extract_all_features:n {#1}
 \@@_preparse_features:

 \@@_load_font:
 \@@_set_scriptlang:
 \@@_get_features:Nn \l_@@_rawfeatures_sclist {}
 \bool_set_false:N \l_@@_firsttime_bool

 \@@_save_family:nTF {#2}
  {
   \@@_save_fontinfo:
   \@@_find_autofonts:
   \DeclareFontFamily{\g_fontspec_encoding_tl}{\l_fontspec_family_tl}{}
   \@@_set_faces:
   \@@_info:nxx {defining-font} {#1} {#2}
⟨*debug⟩
   \typeout{"\l_@@_fontid_tl"~ defined.}
   \@@_warning:nxx {defining-font} {#1} {#2}
⟨/debug⟩
  }
  {
⟨*debug⟩
   \typeout{"\l_@@_fontid_tl"~ already~ defined~ apparently.}
⟨/debug⟩
  }
 \group_end:
}
```

`\@@_load_external_fontoptions:Nn` Load a possible `.fontspec` font configuration file. This file could set font-specific options for the font about to be loaded.

```
\cs_new:Nn \@@_load_external_fontoptions:Nn
{
 \@@_sanitise_fontname:Nn #1 {#2}
```

```
  \tl_set:Nx \l_@@_ext_filename_tl {#1.fontspec}
  \tl_remove_all:Nn \l_@@_ext_filename_tl {~}
  \prop_if_in:NVF \g_@@_fontopts_prop #1
   {
    \exp_args:No \file_if_exist:nT { \l_@@_ext_filename_tl }
     { \file_input:n { \l_@@_ext_filename_tl } }
   }
}
```

`\@@_extract_features:`

```
\cs_new:Nn \@@_extract_all_features:n
{
 \bool_if:NTF \l_@@_disable_defaults_bool
  {
   \clist_set:Nx \l_@@_all_features_clist {#1}
  }
  {
  \prop_get:NVNF \g_@@_fontopts_prop \l_fontspec_fontname_tl \l_@@_fontopts_clist
    { \clist_clear:N \l_@@_fontopts_clist }

  \prop_get:NVNF \g_@@_fontopts_prop \l_@@_family_label_tl \l_@@_family_fontopts_clist
    { \clist_clear:N \l_@@_family_fontopts_clist }
   \tl_clear:N \l_@@_family_label_tl

   \clist_set:Nx \l_@@_all_features_clist
    {
     \g_@@_default_fontopts_clist,
     \l_@@_family_fontopts_clist,
     \l_@@_fontopts_clist,
     #1
    }
  }
 \tl_set:Nx \l_@@_fontid_tl { \tl_to_str:N \l_fontspec_fontname_tl-:-\tl_to_str:N \l_@@_all_feature
⟨*debug⟩
  \typeout{fontid: \l_@@_fontid_tl}
⟨/debug⟩
 }
```

`\@@_preparse_features:` #1 : feature options
#2 : font name

Perform the (multi-step) feature parsing process.

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.

```
\cs_new:Nn \@@_preparse_features:
{
```

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

```
  \@@_if_detect_external:VT \l_@@_basename_tl
   { \keys_set:nn {fontspec-preparse-external} {ExternalLocation} }

```

```
  \keys_set_known:nxN {fontspec-preparse-external}
   { \l_@@_all_features_clist }
   \l_@@_keys_leftover_clist
```

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

```
  \tl_set_eq:NN \l_fontspec_fontname_tl \l_fontspec_fontname_up_tl
  \keys_set_known:nxN {fontspec-renderer} {\l_@@_keys_leftover_clist}
    \l_@@_keys_leftover_clist
  \keys_set_known:nxN {fontspec-preparse} {\l_@@_keys_leftover_clist}
    \l_@@_fontfeat_clist
}
```

`\@@_load_font:`

```
\cs_new:Nn \@@_load_font:
{
 \@@_font_set:Nnn    \l_fontspec_font
    { \@@_fullname:n {\l_fontspec_fontname_up_tl} } {\f@size pt}
 \@@_font_if_null:NT \l_fontspec_font { \@@_error:nx {font-not-found} {\l_fontspec_fontname_up_tl}}
 \@@_set_font_type:
 \@@_font_gset:Nnn   \l_fontspec_font
    { \@@_fullname:n {\l_fontspec_fontname_up_tl} } {\f@size pt}
 \l_fontspec_font % this is necessary for LuaLaTeX to check the scripts properly
}
```

`\@@_if_detect_external:nnT` Check if either the fontname ends with a known font extension.

```
\prg_new_conditional:Nnn \@@_if_detect_external:n {T}
{
 \clist_map_inline:Nn \l_@@_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:
}
\cs_generate_variant:Nn \@@_if_detect_external:nT {V}
```

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

```
\cs_set:Nn \@@_fullname:n
{
 \@@_namewrap:n { #1 \l_@@_extension_tl }
 \l_fontspec_renderer_tl
 \l_@@_optical_size_tl
}
```

`\@@_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:Nn \@@_set_scriptlang:
{
 \bool_if:NT \l_@@_firsttime_bool
```

```
  {
   \tl_if_empty:NTF \l_@@_script_name_tl
    {
     \fontspec_check_script:nTF {latn}
      {
       \tl_set:Nn \l_@@_script_name_tl {Latin}
       \tl_if_empty:NT \l_@@_lang_name_tl
        {
         \tl_set:Nn \l_@@_lang_name_tl {Default}
        }
       \keys_set:nx {fontspec} {Script=\l_@@_script_name_tl}
       \keys_set:nx {fontspec} {Language=\l_@@_lang_name_tl}
      }
      {
       \@@_info:n {no-scripts}
      }
    }
    {
     \tl_if_empty:NT \l_@@_lang_name_tl
      {
       \tl_set:Nn \l_@@_lang_name_tl {Default}
      }
     \keys_set:nx {fontspec} {Script=\l_@@_script_name_tl}
     \keys_set:nx {fontspec} {Language=\l_@@_lang_name_tl}
    }
  }
}
```

`\@@_save_family:nTF` 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.

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.

```
\prg_new_conditional:Nnn \@@_save_family:n {TF}
{
⟨debug⟩\typeout{save~ family:~ #1}
 \cs_if_exist:NT \l_@@_nfss_fam_tl
  {
   \cs_set_eq:cN {g_@@_UID_\l_@@_fontid_tl} \l_@@_nfss_fam_tl
  }
 \cs_if_exist:cF {g_@@_UID_\l_@@_fontid_tl}
  {
  % The font name is fully expanded, in case it's defined in terms of macros, before having its space
   \tl_set:Nx \l_@@_tmp_tl {#1}
   \tl_remove_all:Nn \l_@@_tmp_tl {~}

   \cs_if_exist:cTF {g_@@_family_ \l_@@_tmp_tl _int}
    { \int_gincr:c  {g_@@_family_ \l_@@_tmp_tl _int} }
    { \int_new:c    {g_@@_family_ \l_@@_tmp_tl _int} }

   \tl_gset:cx {g_@@_UID_\l_@@_fontid_tl}
    {
```

```
      \l_@@_tmp_tl ( \int_use:c {g_@@_family_ \l_@@_tmp_tl _int} )
     }
   }
  \tl_gset:Nv \l_fontspec_family_tl {g_@@_UID_\l_@@_fontid_tl}
  \cs_if_exist:cTF {g_@@_ \l_fontspec_family_tl _prop}
    \prg_return_false: \prg_return_true:
 }
```

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

```
\cs_generate_variant:Nn \prop_gput:Nnn {cnV}
\cs_generate_variant:Nn \prop_gput:Nnn {cnx}
\cs_new:Nn \@@_save_fontinfo:
 {
  \prop_new:c {g_@@_ \l_fontspec_family_tl _prop}
  \prop_gput:cnx {g_@@_ \l_fontspec_family_tl _prop} {fontname} { \l_@@_basename_tl }
  \prop_gput:cnx {g_@@_ \l_fontspec_family_tl _prop} {options}  { \l_@@_all_features_clist }
  \prop_gput:cnx {g_@@_ \l_fontspec_family_tl _prop} {fontdef}
   {
    \@@_fullname:n {\l_fontspec_fontname_tl} :
    \l_@@_pre_feat_sclist \l_@@_rawfeatures_sclist
   }
  \prop_gput:cnV {g_@@_ \l_fontspec_family_tl _prop} {script-num} \l_fontspec_script_int
  \prop_gput:cnV {g_@@_ \l_fontspec_family_tl _prop} {lang-num} \l_fontspec_language_int
  \prop_gput:cnV {g_@@_ \l_fontspec_family_tl _prop} {script-tag} \l_fontspec_script_tl
  \prop_gput:cnV {g_@@_ \l_fontspec_family_tl _prop} {lang-tag} \l_fontspec_lang_tl

 }
```

23.5.1 Setting font shapes in a family

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

The combination shapes are searched first because they use information that may be redefined in the single cases. E.g., if no bold font is specified then `set_autofont` will attempt to set it. This has subtle/small ramifications on the logic of choosing the bold italic font.

`\@@_find_autofonts:`

```
\cs_new:Nn \@@_find_autofonts:
 {
  \bool_if:nF {\l_@@_noit_bool || \l_@@_nobf_bool}
   {
   \@@_set_autofont:Nnn \l_fontspec_fontname_bfit_tl {\l_fontspec_fontname_it_tl} {/B}
   \@@_set_autofont:Nnn \l_fontspec_fontname_bfit_tl {\l_fontspec_fontname_bf_tl} {/I}
   \@@_set_autofont:Nnn \l_fontspec_fontname_bfit_tl {\l_fontspec_fontname_tl} {/BI}
   }

  \bool_if:NF \l_@@_nobf_bool
   {
   \@@_set_autofont:Nnn \l_fontspec_fontname_bf_tl {\l_fontspec_fontname_tl} {/B}
   }

```

```
  \bool_if:NF \l_@@_noit_bool
   {
   \@@_set_autofont:Nnn \l_fontspec_fontname_it_tl {\l_fontspec_fontname_tl} {/I}
   }

 \@@_set_autofont:Nnn \l_fontspec_fontname_bfsl_tl {\l_fontspec_fontname_sl_tl} {/B}
}
```

`\@@_set_faces:`

```
\cs_new:Nn \@@_set_faces:
{
 \@@_add_nfssfont:oooo \mddefault \updefault \l_fontspec_fontname_tl     \l_@@_fontfeat_up_clist
 \@@_add_nfssfont:oooo \bfdefault \updefault \l_fontspec_fontname_bf_tl   \l_@@_fontfeat_bf_clist
 \@@_add_nfssfont:oooo \mddefault \itdefault \l_fontspec_fontname_it_tl   \l_@@_fontfeat_it_clist
 \@@_add_nfssfont:oooo \mddefault \sldefault \l_fontspec_fontname_sl_tl   \l_@@_fontfeat_sl_clist
 \@@_add_nfssfont:oooo \bfdefault \itdefault \l_fontspec_fontname_bfit_tl \l_@@_fontfeat_bfit_clist
 \@@_add_nfssfont:oooo \bfdefault \sldefault \l_fontspec_fontname_bfsl_tl \l_@@_fontfeat_bfsl_clist

  \prop_map_inline:Nn \l_@@_nfssfont_prop { \@@_set_faces_aux:nnnnn ##2 }
}
\cs_new:Nn \@@_set_faces_aux:nnnnn
{
  \fontspec_complete_fontname:Nn \l_@@_curr_fontname_tl {#3}
  \@@_make_font_shapes:Nnnnn \l_@@_curr_fontname_tl {#1} {#2} {#4} {#5}
}
```

23.5.2 Fonts

`\@@_set_font_type:`

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

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

```
\cs_new:Nn \@@_set_font_type:
⟨*xetexx⟩
{
  \bool_set_false:N \l_@@_tfm_bool
  \bool_set_false:N \l_@@_atsui_bool
  \bool_set_false:N \l_@@_ot_bool
  \bool_set_false:N \l_@@_mm_bool
  \bool_set_false:N \l_@@_graphite_bool
  \ifcase\XeTeXfonttype\l_fontspec_font
    \bool_set_true:N \l_@@_tfm_bool
  \or
    \bool_set_true:N \l_@@_atsui_bool
    \ifnum\XeTeXcountvariations\l_fontspec_font > \c_zero
      \bool_set_true:N \l_@@_mm_bool
    \fi
  \or
    \bool_set_true:N \l_@@_ot_bool
  \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
 {
  \bool_if:NTF \l_@@_atsui_bool
   { \tl_set:Nn \l_fontspec_renderer_tl {/AAT} }
   {
     \bool_if:NT \l_@@_ot_bool
      { \tl_set:Nn \l_fontspec_renderer_tl {/OT} }
   }
 }
```

⟨/xetexx⟩

⟨*luatex⟩

```
{
 \bool_set_true:N \l_@@_ot_bool
}
```

⟨/luatex⟩

`\@@_set_autofont:Nnn` `#1` : Font name tl
`#2` : Base font name
`#3` : Font name modifier

This function looks for font with ⟨*name*⟩ and ⟨*modifier*⟩ `#2#3`, and if found (i.e., different to font with name `#2`) stores it in tl `#1`. A modifier is something like `/B` to look for a bold font, for example.

We can't match external fonts in this way (in XeTeX anyway; todo: test with LuaTeX). If ⟨*font name tl*⟩ is not empty, then it's already been specified by the user so abort. If ⟨*Base font name*⟩ is not given, we also abort for obvious reasons.

If ⟨*font name tl*⟩ is empty, then proceed. If not found, ⟨*font name tl*⟩ remains empty. Otherwise, we have a match.

```
\cs_generate_variant:Nn \tl_if_empty:nF {x}
\cs_new:Nn \@@_set_autofont:Nnn
{
 \bool_if:NF \l_@@_external_bool
  {
 \tl_if_empty:xF {#2}
  {
   \tl_if_empty:NT #1
    {
     \@@_if_autofont:nnTF {#2} {#3}
      { \tl_set:Nx #1 {#2#3} }
      { \@@_info:nx {no-font-shape} {#2#3} }
    }
  }
  }
}

\prg_new_conditional:Nnn \@@_if_autofont:nn {T,TF}
{
 \@@_font_set:Nnn \l_tmpa_font { \@@_fullname:n {#1}   } {\f@size pt}
```

```
  \@@_font_set:Nnn \l_tmpb_font { \@@_fullname:n {#1#2} } {\f@size pt}
  \str_if_eq_x:nnTF { \fontname \l_tmpa_font } { \fontname \l_tmpb_font }
   { \prg_return_false: }
   { \prg_return_true: }
}
```

`\@@_make_font_shapes:Nnnnn` #1 : Font name
#2 : Font series
#3 : Font shape
#4 : Font features
#5 : Size 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.

Next, the small caps are defined. [...] is used to define the appropriate string for activating small caps in the font, if they exist. If we are defining small caps for the upright shape, then the small caps shape default is used. For an *italic* font, however, the shape parameter is overloaded and we must call italic small caps by their own identifier. See Section 23.7 on page 111 for the code that enables this usage.

```
\cs_new:Nn \@@_make_font_shapes:Nnnnn
{
  \group_begin:
    \@@_load_fontname:n {#1}
    \@@_declare_shape:nnxx {#2} {#3} { \l_@@_fontopts_clist, #4 } {#5}
  \group_end:
}

\cs_new:Nn \@@_load_fontname:n
{
    \@@_load_external_fontoptions:Nn \l_fontspec_fontname_tl {#1}
   \prop_get:NVNF \g_@@_fontopts_prop \l_fontspec_fontname_tl \l_@@_fontopts_clist
     { \clist_clear:N \l_@@_fontopts_clist }
   \@@_font_set:Nnn \l_fontspec_font {\@@_fullname:n {\l_fontspec_fontname_tl}} {\f@size pt}
   \@@_font_if_null:NT \l_fontspec_font { \@@_error:nx {font-not-found} {#1} }
}
```

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

`\@@_declare_shape:nnnn` #1 : Font series
#2 : Font shape
#3 : Font features
#4 : Size features

Wrapper for `\DeclareFontShape`. And finally the actual font shape declaration using `\l_@@_nfss_tl` defined above. `\l_@@_postadjust_tl` is defined in various places to deal with things like the hyphenation character and interword spacing.

The main part is to loop through `SizeFeatures` arguments, which are of the form

`SizeFeatures={{<one>},{<two>},{<three>}}`.

```
\cs_new:Nn \@@_declare_shape:nnnn
{
```

```
  \tl_clear:N \l_@@_nfss_tl
  \tl_clear:N \l_@@_nfss_sc_tl
  \tl_set_eq:NN \l_@@_saved_fontname_tl \l_fontspec_fontname_tl

  \exp_args:Nx \clist_map_inline:nn {#4}
   {
    \tl_clear:N \l_@@_size_tl
    \tl_set_eq:NN \l_@@_sizedfont_tl \l_@@_saved_fontname_tl % in case not spec'ed

    \keys_set_known:nxN {fontspec-sizing} { \exp_after:wN \use:n ##1 }
      \l_@@_sizing_leftover_clist
    \tl_if_empty:NT \l_@@_size_tl { \@@_error:n {no-size-info} }

    % "normal"
    \@@_load_fontname:n {\l_@@_sizedfont_tl}
    \@@_setup_nfss:Nnn \l_@@_nfss_tl {#3} {}

    % small caps
    \clist_set_eq:NN \l_@@_fontfeat_curr_clist \l_@@_fontfeat_sc_clist

    \bool_if:NF \l_@@_nosc_bool
     {
      \tl_if_empty:NTF \l_fontspec_fontname_sc_tl
       {
⟨debug⟩       \typeout{Attempting~ small~ caps?}
        \@@_make_smallcaps:TF
         {
⟨debug⟩       \typeout{Small~ caps~ found.}
          \clist_put_left:Nn \l_@@_fontfeat_curr_clist {Letters=SmallCaps}
         }
         {
⟨debug⟩       \typeout{Small~ caps~ not~ found.}
          \bool_set_true:N \l_@@_nosc_bool
         }
       }
      { \@@_load_fontname:n {\l_fontspec_fontname_sc_tl} }% local for each size
     }

    \bool_if:NF \l_@@_nosc_bool
     {
      \@@_setup_nfss:Nnn \l_@@_nfss_sc_tl {#3} {\l_@@_fontfeat_curr_clist}
     }

   }

  \@@_declare_shapes_normal:nn  {#1} {#2}
  \@@_declare_shape_slanted:nn  {#1} {#2}
  \@@_declare_shape_loginfo:nnn {#1} {#2} {#3}
 }
\cs_generate_variant:Nn \@@_declare_shape:nnnn {nnxx}

```

```
\cs_new:Nn \@@_setup_nfss:Nnn
{
 \@@_get_features:Nn \l_@@_rawfeatures_sclist
  { #2 , \l_@@_sizing_leftover_clist , #3 }

 \tl_put_right:Nx #1
  {
   <\l_@@_size_tl> \l_@@_scale_tl
   \@@_fontwrap:n
    {
     \@@_fullname:n { \l_fontspec_fontname_tl }
     : \l_@@_pre_feat_sclist \l_@@_rawfeatures_sclist
    }
  }
}

\cs_new:Nn \@@_declare_shapes_normal:nn
{
  \@@_DeclareFontShape:xxxxxx {\g_fontspec_encoding_tl} {\l_fontspec_family_tl}
     {#1} {#2} {\l_@@_nfss_tl}{\l_@@_postadjust_tl}

   \bool_if:NF \l_@@_nosc_bool
    {
    \@@_DeclareFontShape:xxxxxx {\g_fontspec_encoding_tl} {\l_fontspec_family_tl}
       {#1}
       {\str_if_eq_x:nnTF {#2} {\itdefault} \sidefault \scdefault}
       {\l_@@_nfss_sc_tl}{\l_@@_postadjust_tl}
    }
}

\cs_new:Nn \@@_DeclareFontShape:nnnnnn
{
 \group_begin:
   \normalsize
   \cs_undefine:c {#1/#2/#3/#4/\f@size}
 \group_end:
 \DeclareFontShape{#1}{#2}{#3}{#4}{#5}{#6}
}
\cs_generate_variant:Nn \@@_DeclareFontShape:nnnnnn {xxxxxx}
```

This extra stuff for the slanted shape substitution is a little bit awkward. We define the slanted shape to be a synonym for it when (a) we're defining an italic font, but also (b) when the default slanted shape isn't 'it'. (Presumably this turned up once in a test and I realised it caused problems. I doubt this would happen much.)

We should test when a slanted font has been specified and not run this code if so, but the `\@@_set_slanted:` code will overwrite this anyway if necessary.

```
\cs_new:Nn \@@_declare_shape_slanted:nn
{
 \bool_if:nT
  {
    \str_if_eq_x_p:nn {#2} {\itdefault}  &&
   !(\str_if_eq_x_p:nn {\itdefault} {\sldefault})
```

```
  }
  {
  \@@_DeclareFontShape:xxxxxx {\g_fontspec_encoding_tl}{\l_fontspec_family_tl}{#1}{\sldefault}
     {<->ssub*\l_fontspec_family_tl/#1/\itdefault}{\l_@@_postadjust_tl}
  }
}
```

Lastly some informative messaging.

```
\cs_new:Nn \@@_declare_shape_loginfo:nnn
{
 \tl_gput_right:Nx \l_fontspec_defined_shapes_tl
  {
   \exp_not:n { \\ \\ }
   *~ '\exp_not:N \str_case:nnn {#1/#2}
    {
      {\mddefault/\updefault} {normal}
      {\bfdefault/\updefault} {bold}
      {\mddefault/\itdefault} {italic}
      {\bfdefault/\itdefault} {bold~ italic}
    } {#2/#3}'~
   with~ NFSS~ spec.: \exp_not:N \\
   \l_@@_nfss_tl
   \exp_not:n { \\ \\ }
   *~ '\exp_not:N \str_case:nnn {#1/\scdefault}
    {
      {\mddefault/\scdefault} {small~ caps}
      {\bfdefault/\scdefault} {bold~ small~ caps}
      {\mddefault/\sidefault} {italic~ small~ caps}
      {\bfdefault/\sidefault} {bold~ italic~ small~ caps}
    } {#2/#3}'~
   with~ NFSS~ spec.: \exp_not:N \\
   \l_@@_nfss_sc_tl
   \tl_if_empty:NF \l_@@_postadjust_tl
    {
    \exp_not:N \\ and~ font~ adjustment~ code: \exp_not:N \\ \l_@@_postadjust_tl
    }
  }
}
```

`\l_@@_pre_feat_sclist` These are the features always applied to a font selection before other features.

```
\clist_set:Nn \l_@@_pre_feat_sclist
⟨*xetexx⟩
{
 \bool_if:NT \l_@@_ot_bool
  {
   \tl_if_empty:NF \l_fontspec_script_tl
    {
     script   = \l_fontspec_script_tl ;
     language = \l_fontspec_lang_tl   ;
    }
  }
}
```

```
⟨/xetexx⟩
⟨*luatex⟩
{
 mode     = \l_fontspec_mode_tl   ;
 \tl_if_empty:NF \l_fontspec_script_tl
  {
   script   = \l_fontspec_script_tl ;
   language = \l_fontspec_lang_tl   ;
  }
}
⟨/luatex⟩
```

23.5.3 Features

`\@@_get_features:Nn` This macro is a wrapper for `\keys_set:nn` 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. Its argument is any additional features to prepend to the default.

```
\cs_set:Nn \@@_get_features:Nn
{
 \sclist_clear:N \l_@@_rawfeatures_sclist
 \tl_clear:N \l_@@_scale_tl
 \tl_set_eq:NN \l_@@_opacity_tl \g_@@_opacity_tl
 \tl_set_eq:NN \l_@@_hexcol_tl \g_@@_hexcol_tl
 \tl_set_eq:NN \l_@@_postadjust_tl \g_@@_postadjust_tl
 \tl_clear:N \l_@@_wordspace_adjust_tl
 \tl_clear:N \l_@@_punctspace_adjust_tl

 \keys_set_known:nxN {fontspec-renderer} {\l_@@_fontfeat_clist,#2}
   \l_@@_keys_leftover_clist
 \keys_set:nx {fontspec} {\l_@@_keys_leftover_clist}
```

Finish the colour specification. Do not set the colour if not explicitly spec'd else `\color` (using specials) will not work.

```
 \str_if_eq_x:nnF { \l_@@_hexcol_tl \l_@@_opacity_tl }
                  { \g_@@_hexcol_tl \g_@@_opacity_tl }
  {
   \@@_update_featstr:n { color = \l_@@_hexcol_tl\l_@@_opacity_tl }
  }

 \tl_set_eq:NN #1 \l_@@_rawfeatures_sclist
}
```

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

```
\tl_clear:N \l_@@_family_label_tl
\tl_clear:N \l_fontspec_fontname_bf_tl
\tl_clear:N \l_fontspec_fontname_it_tl
\tl_clear:N \l_fontspec_fake_slant_tl
\tl_clear:N \l_fontspec_fake_embolden_tl
\tl_clear:N \l_fontspec_fontname_bfit_tl
\tl_clear:N \l_fontspec_fontname_sl_tl
\tl_clear:N \l_fontspec_fontname_bfsl_tl
```

```
\tl_clear:N \l_fontspec_fontname_sc_tl
\tl_clear:N \l_@@_fontfeat_up_clist
\tl_clear:N \l_@@_fontfeat_bf_clist
\tl_clear:N \l_@@_fontfeat_it_clist
\tl_clear:N \l_@@_fontfeat_bfit_clist
\tl_clear:N \l_@@_fontfeat_sl_clist
\tl_clear:N \l_@@_fontfeat_bfsl_clist
\tl_clear:N \l_@@_fontfeat_sc_clist
\tl_clear:N \l_@@_script_name_tl
\tl_clear:N \l_fontspec_script_tl
\tl_clear:N \l_@@_lang_name_tl
\tl_clear:N \l_fontspec_lang_tl
\tl_set:Nn  \g_@@_postadjust_tl { \l_@@_wordspace_adjust_tl \l_@@_punctspace_adjust_tl }

\clist_set:Nn \l_@@_sizefeat_clist {Size={-}}
\tl_new:N  \g_@@_hexcol_tl
\tl_new:N  \g_@@_opacity_tl
\tl_set:Nn \g_@@_hexcol_tl {000000}
\tl_set:Nn \g_@@_opacity_tl {FF~}
```

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

```
\cs_set:Npn \@@_init:
 {
  \bool_set_false:N \l_@@_ot_bool
  \bool_set_true:N \l_@@_firsttime_bool
  \cs_set:Npn \@@_namewrap:n ##1 { ##1 }
  \tl_clear:N \l_@@_optical_size_tl
  \tl_clear:N \l_fontspec_renderer_tl
  \tl_clear:N \l_fontspec_defined_shapes_tl
  \tl_clear:N \g_@@_curr_series_tl

  % This is for detecting font families when assigning default features.
 % Replace defaults for the standard families because they're not set in the usual way:
  \exp_args:NV \str_case:nnn {\l_@@_family_label_tl}
   {
    {\rmdefault} { \tl_set:Nn \l_@@_family_label_tl {\g_@@_rmfamily_family} }
    {\sfdefault} { \tl_set:Nn \l_@@_family_label_tl {\g_@@_sffamily_family} }
    {\ttdefault} { \tl_set:Nn \l_@@_family_label_tl {\g_@@_ttfamily_family} }
   }{}

⟨*luatex⟩
  \tl_set:Nn \l_fontspec_mode_tl {node}
  \luatexprehyphenchar   =`\-  % fixme
  \luatexposthyphenchar  = 0   % fixme
  \luatexpreexhyphenchar = 0   % fixme
  \luatexpostexhyphenchar= 0   % fixme
⟨/luatex⟩
 }
```

`\@@_make_smallcaps:TF` This macro checks if the font contains small caps.

```
\cs_set:Nn \fontspec_make_ot_smallcaps:TF
```

```
 {
  \fontspec_check_ot_feat:nTF {+smcp} {#1} {#2}
 }
⟨*xetexx⟩
\cs_set:Nn \@@_make_smallcaps:TF
 {
  \bool_if:NTF \l_@@_ot_bool
   { \fontspec_make_ot_smallcaps:TF {#1} {#2} }
   {
     \bool_if:NT \l_@@_atsui_bool
      { \fontspec_make_AAT_feature_string:nnTF {3}{3} {#1} {#2} }
   }
 }
⟨/xetexx⟩
⟨*luatex⟩
\cs_set_eq:NN \@@_make_smallcaps:TF \fontspec_make_ot_smallcaps:TF
⟨/luatex⟩
```

`\sclist_put_right:Nn` I'm hardly going to write an 'sclist' module but a couple of functions are useful. Here, items in semi-colon lists are always followed by a semi-colon (as opposed to the s.-c's being placed between elements) so we can append sclists without worrying about it.

```
\cs_set_eq:NN \sclist_clear:N \tl_clear:N
\cs_new:Nn \sclist_gput_right:Nn
 { \tl_gput_right:Nn #1 {#2;} }
\cs_generate_variant:Nn \sclist_gput_right:Nn {Nx}
```

`\@@_update_featstr:n` `\l_@@_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.

```
\cs_new:Nn \@@_update_featstr:n
 {
  \bool_if:NF \l_@@_firsttime_bool
   {
    \sclist_gput_right:Nx \l_@@_rawfeatures_sclist {#1}
   }
 }
```

`\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:Nn \fontspec_make_feature:nnn
⟨*xetexx⟩
 {
  \bool_if:NTF \l_@@_ot_bool
   { \fontspec_make_OT_feature:n {#3} }
   {
     \bool_if:NT \l_@@_atsui_bool
      { \fontspec_make_AAT_feature:nn {#1}{#2} }
   }
 }
⟨/xetexx⟩
⟨*luatex⟩
 { \fontspec_make_OT_feature:n {#3} }
```

```
⟨/luatex⟩
\cs_generate_variant:Nn \fontspec_make_feature:nnn {nnx}
\cs_new:Nn \fontspec_make_AAT_feature:nn
 {
  \tl_if_empty:nTF {#1}
   { \@@_warning:n {aat-feature-not-exist} }
   {
     \fontspec_make_AAT_feature_string:nnTF {#1}{#2}
      {
       \@@_update_featstr:n {\l_fontspec_feature_string_tl}
     }
     { \@@_warning:nx {aat-feature-not-exist-in-font} {#1,#2} }
   }
 }
\cs_new:Nn \fontspec_make_OT_feature:n
 {
  \tl_if_empty:nTF {#1}
   { \@@_warning:n {icu-feature-not-exist} }
   {
     \fontspec_check_ot_feat:nTF {#1}
      {
       \@@_update_featstr:n {#1}
      }
      { \@@_warning:nx {icu-feature-not-exist-in-font} {#1} }
   }
 }
\cs_new_protected:Nn \fontspec_make_numbered_feature:nn
 {
  \fontspec_check_ot_feat:nTF {#1}
   {
    \@@_update_featstr:n { #1 = #2 }
   }
   { \@@_warning:nx {icu-feature-not-exist-in-font} {#1} }
 }
\cs_generate_variant:Nn \fontspec_make_numbered_feature:nn {xn}
```

`\@@_define_font_feature:n`
`\@@_define_feature_option:nnnnn`
`\fontspec_define_numbered_feat:nnnn`

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

```
\cs_new:Nn \@@_define_font_feature:n
 {
  \keys_define:nn {fontspec} { #1 .multichoice: }
 }
\cs_new:Nn \@@_define_feature_option:nnnnn
 {
  \keys_define:nn {fontspec}
   {
    #1/#2 .code:n = { \fontspec_make_feature:nnn{#3}{#4}{#5} }
   }
 }
\cs_new:Nn \fontspec_define_numbered_feat:nnnn
 {
  \keys_define:nn {fontspec}
```

```
  {
   #1/#2 .code:n =
     { \fontspec_make_numbered_feature:nn {#3}{#4} }
  }
}
```

`ec_make_AAT_feature_string:nnTF` 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 [...], but also used to check if small caps exists in the requested font (see page 81).

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

Finally, save out the complete feature string in `\l_fontspec_feature_string_tl`.

```
\prg_new_conditional:Nnn \fontspec_make_AAT_feature_string:nn {TF,T,F}
{
 \tl_set:Nx \l_tmpa_tl { \XeTeXfeaturename \l_fontspec_font #1 }
 \tl_if_empty:NTF \l_tmpa_tl
  { \prg_return_false: }
  {
   \int_compare:nTF { \XeTeXisexclusivefeature\l_fontspec_font #1 > 0 }
    {
     \tl_set:Nx \l_tmpb_tl {\XeTeXselectorname\l_fontspec_font #1\space #2}
    }
    {
     \int_if_even:nTF {#2}
      {
      \tl_set:Nx \l_tmpb_tl {\XeTeXselectorname\l_fontspec_font #1\space #2}
      }
      {
       \tl_set:Nx \l_tmpb_tl
        {
         \XeTeXselectorname\l_fontspec_font #1\space \numexpr#2-1\relax
        }
       \tl_if_empty:NF \l_tmpb_tl { \tl_put_left:Nn \l_tmpb_tl {!} }
      }
    }
   \tl_if_empty:NTF \l_tmpb_tl
    { \prg_return_false: }
    {
     \tl_set:Nx \l_fontspec_feature_string_tl { \l_tmpa_tl = \l_tmpb_tl }
     \prg_return_true:
    }
  }
}
```

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

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

```
\cs_set:Nn \fontspec_iv_str_to_num:Nn
 {
  \fontspec_iv_str_to_num:w #1 \q_nil #2 \c_empty_tl \c_empty_tl \q_nil
 }
\cs_set:Npn \fontspec_iv_str_to_num:w #1 \q_nil #2#3#4#5#6 \q_nil
 {
  \int_set:Nn #1
   {
      `#2 * "1000000
    + `#3 * "10000
    + \ifx \c_empty_tl #4 32 \else `#4 \fi * "100
    + \ifx \c_empty_tl #5 32 \else `#5 \fi
   }
 }
\cs_generate_variant:Nn \fontspec_iv_str_to_num:Nn {No}
\cs_set:Nn \fontspec_v_str_to_num:Nn
 {
  \bool_if:nTF
   {
    \tl_if_head_eq_charcode_p:nN {#2} {+} ||
    \tl_if_head_eq_charcode_p:nN {#2} {-}
   }
   { \fontspec_iv_str_to_num:No #1 { \use_none:n #2 } }
   { \fontspec_iv_str_to_num:Nn #1 {#2} }
 }
```

\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.

```
\prg_new_conditional:Nnn \fontspec_check_script:n {TF}
⟨*xetexx⟩
 {
  \fontspec_iv_str_to_num:Nn \l_fontspec_strnum_int {#1}
  \int_set:Nn \l_tmpb_int { \XeTeXOTcountscripts \l_fontspec_font }
  \int_zero:N \l_tmpa_int
  \@tempswafalse
  \bool_until_do:nn { \int_compare_p:nNn \l_tmpa_int = \l_tmpb_int }
   {
   \ifnum \XeTeXOTscripttag\l_fontspec_font \l_tmpa_int = \l_fontspec_strnum_int
      \@tempswatrue
      \int_set:Nn \l_tmpa_int {\l_tmpb_int}
    \else
      \int_incr:N \l_tmpa_int
```

```
    \fi
   }
  \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
 }
⟨/xetexx⟩
⟨*luatex⟩
 {
  \directlua{fontspec.check_ot_script("l_fontspec_font", "#1")}
  \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
 }
⟨/luatex⟩
```

`\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 `\l_fontspec_script_int`. By default, that's the number corresponding to '`latn`'.

```
\prg_new_conditional:Nnn \fontspec_check_lang:n {TF}
⟨*xetexx⟩
 {
  \fontspec_iv_str_to_num:Nn \l_fontspec_strnum_int {#1}
  \int_set:Nn \l_tmpb_int
   { \XeTeXOTcountlanguages \l_fontspec_font \l_fontspec_script_int }
  \int_zero:N \l_tmpa_int
  \@tempswafalse
  \bool_until_do:nn { \int_compare_p:nNn \l_tmpa_int = \l_tmpb_int }
   {
   \ifnum\XeTeXOTlanguagetag\l_fontspec_font\l_fontspec_script_int \l_tmpa_int =\l_fontspec_strnum
      \@tempswatrue
      \int_set:Nn \l_tmpa_int {\l_tmpb_int}
    \else
      \int_incr:N \l_tmpa_int
    \fi
   }
  \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
 }
⟨/xetexx⟩
⟨*luatex⟩
 {
  \directlua
   {
   fontspec.check_ot_lang( "l_fontspec_font", "#1", "\l_fontspec_script_tl" )
   }
  \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
 }
⟨/luatex⟩
```

`\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 `\l_fontspec_script_int`. By default, that's the number corresponding to '`latn`'. The language used is `\l_fontspec_language_int`, by default `0`, the 'default language'.

```
\prg_new_conditional:Nnn \fontspec_check_ot_feat:n {TF,T}
```

```
⟨*xetexx⟩
{
 \int_set:Nn \l_tmpb_int
  {
   \XeTeXOTcountfeatures \l_fontspec_font
                         \l_fontspec_script_int
                         \l_fontspec_language_int
  }
 \fontspec_v_str_to_num:Nn \l_fontspec_strnum_int {#1}
 \int_zero:N \l_tmpa_int
 \@tempswafalse
 \bool_until_do:nn { \int_compare_p:nNn \l_tmpa_int = \l_tmpb_int }
  {
  \ifnum\XeTeXOTfeaturetag\l_fontspec_font\l_fontspec_script_int\l_fontspec_language_int
        \l_tmpa_int =\l_fontspec_strnum_int
     \@tempswatrue
     \int_set:Nn \l_tmpa_int {\l_tmpb_int}
   \else
     \int_incr:N \l_tmpa_int
   \fi
  }
 \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
}
⟨/xetexx⟩
⟨*luatex⟩
{
 \directlua
  {
   fontspec.check_ot_feat(
                           "l_fontspec_font", "#1",
                           "\l_fontspec_lang_tl", "\l_fontspec_script_tl"
                          )
  }
 \if@tempswa \prg_return_true: \else: \prg_return_false: \fi:
}
⟨/luatex⟩
```

23.6 keyval definitions

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

```
\cs_new:Nn \@@_keys_define_code:nnn
{
 \keys_define:nn {#1} { #2 .code:n = {#3} }
}
```

23.6.1 Pre-parsing naming information

These features are extracted from the font feature list before all others.

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. Other-

wise, the argument given defines the file path of the font.

```
\bool_new:N \l_@@_external_bool
\@@_keys_define_code:nnn {fontspec-preparse-external} {ExternalLocation}
 {
  \bool_set_true:N \l_@@_nobf_bool
  \bool_set_true:N \l_@@_noit_bool
  \bool_set_true:N \l_@@_external_bool
  \cs_gset:Npn \@@_namewrap:n ##1 { [ #1 ##1 ] }
⟨*xetexx⟩
  \keys_set:nn {fontspec-renderer} {Renderer=OpenType}
⟨/xetexx⟩
 }
\aliasfontfeature{ExternalLocation}{Path}
```

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

```
\@@_keys_define_code:nnn {fontspec-preparse-external} {Extension}
 {
  \tl_set:Nn \l_@@_extension_tl {#1}
  \bool_if:NF \l_@@_external_bool
   {
    \keys_set:nn {fontspec-preparse-external} {ExternalLocation}
   }
 }
\tl_clear:N \l_@@_extension_tl
```

23.6.2 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.

```
\keys_define:nn {fontspec-renderer}
 {
  Renderer .choices:nn =
   {AAT,ICU,OpenType,Graphite,Full,Basic}
   {
    \int_compare:nTF {\l_keys_choice_int <= 4} {
⟨*xetexx⟩
      \tl_set:Nv \l_fontspec_renderer_tl
        { g_fontspec_renderer_tag_ \l_keys_choice_tl }
⟨/xetexx⟩
⟨*luatex⟩
      \@@_warning:nx {only-xetex-feature} {Renderer=AAT/OpenType/Graphite}
⟨/luatex⟩
     }
     {
⟨*xetexx⟩
      \@@_warning:nx {only-luatex-feature} {Renderer=Full/Basic}
⟨/xetexx⟩
```

```
⟨*luatex⟩
      \tl_set:Nv \l_fontspec_mode_tl
        { g_fontspec_mode_tag_ \l_keys_choice_tl }
⟨/luatex⟩
    }
  }
}
\tl_set:cn {g_fontspec_renderer_tag_AAT} {/AAT}
\tl_set:cn {g_fontspec_renderer_tag_ICU} {/OT}
\tl_set:cn {g_fontspec_renderer_tag_OpenType} {/OT}
\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.

```
\@@_keys_define_code:nnn {fontspec-preparse} {Script}
{
⟨xetexx⟩    \keys_set:nn {fontspec-renderer} {Renderer=OpenType}
  \tl_set:Nn \l_@@_script_name_tl {#1}
}
```

Exactly the same:

```
\@@_keys_define_code:nnn {fontspec-preparse} {Language}
{
⟨xetexx⟩    \keys_set:nn {fontspec-renderer} {Renderer=OpenType}
  \tl_set:Nn \l_@@_lang_name_tl {#1}
}
```

23.6.3 Bold/italic choosing options

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

Bold (NFSS) Series By default, fontspec uses the default bold series, `\bfdefault`. We want to be able to make this extensible.

```
\seq_new:N \g_@@_bf_series_seq
\@@_keys_define_code:nnn {fontspec-preparse-external} {BoldSeries}
{
  \tl_gset:Nx \g_@@_curr_series_tl { #1 }
  \seq_gput_right:Nx \g_@@_bf_series_seq { #1 }
}
```

Fonts Upright:

```
\@@_keys_define_code:nnn {fontspec-preparse-external} {UprightFont}
{
  \fontspec_complete_fontname:Nn \l_fontspec_fontname_up_tl {#1}
}
\@@_keys_define_code:nnn {fontspec-preparse-external} {FontName}
{
  \fontspec_complete_fontname:Nn \l_fontspec_fontname_up_tl {#1}
}
```

Bold:

```
\cs_generate_variant:Nn \tl_if_eq:nnT {ox}
\cs_generate_variant:Nn \prop_put:Nnn {NxV}
\@@_keys_define_code:nnn {fontspec-preparse-external} {BoldFont}
{
 \tl_if_empty:nTF {#1}
  {
   \bool_set_true:N \l_@@_nobf_bool
  }
  {
   \bool_set_false:N \l_@@_nobf_bool
   \fontspec_complete_fontname:Nn \l_@@_curr_bfname_tl {#1}

   \seq_if_empty:NT \g_@@_bf_series_seq
    {
     \tl_gset:Nx \g_@@_curr_series_tl {\bfdefault}
     \seq_put_right:Nx \g_@@_bf_series_seq {\bfdefault}
    }
   \tl_if_eq:oxT \g_@@_curr_series_tl {\bfdefault}
    { \tl_set_eq:NN \l_fontspec_fontname_bf_tl \l_@@_curr_bfname_tl }

⟨debug⟩\typeout{Setting~bold~font~"\l_@@_curr_bfname_tl"~with~series~"\g_@@_curr_series_tl"}

   \prop_put:NxV \l_@@_nfss_prop
    {BoldFont-\g_@@_curr_series_tl} \l_@@_curr_bfname_tl

  }
}
\prop_new:N \l_@@_nfss_prop
```

Same for italic:

```
\@@_keys_define_code:nnn {fontspec-preparse-external} {ItalicFont}
{
 \tl_if_empty:nTF {#1}
  {
   \bool_set_true:N \l_@@_noit_bool
  }
  {
   \bool_set_false:N \l_@@_noit_bool
   \fontspec_complete_fontname:Nn \l_fontspec_fontname_it_tl {#1}
  }
}
```

Simpler for bold+italic & slanted:

```
\@@_keys_define_code:nnn {fontspec-preparse-external} {BoldItalicFont}
{
 \fontspec_complete_fontname:Nn \l_fontspec_fontname_bfit_tl {#1}
}
\@@_keys_define_code:nnn {fontspec-preparse-external} {SlantedFont}
{
 \fontspec_complete_fontname:Nn \l_fontspec_fontname_sl_tl {#1}
}
```

```
\@@_keys_define_code:nnn {fontspec-preparse-external} {BoldSlantedFont}
{
 \fontspec_complete_fontname:Nn \l_fontspec_fontname_bfsl_tl {#1}
}
```

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

```
\@@_keys_define_code:nnn {fontspec} {SmallCapsFont}
{
 \tl_if_empty:nTF {#1}
  {
   \bool_set_true:N \l_@@_nosc_bool
  }
  {
   \bool_set_false:N \l_@@_nosc_bool
   \fontspec_complete_fontname:Nn \l_fontspec_fontname_sc_tl {#1}
  }
}
```

`\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:Nn \fontspec_complete_fontname:Nn
{
 \tl_set:Nx #1 {#2}
 \tl_replace_all:Nnx #1 {*} {\l_@@_basename_tl}
⟨luatex⟩  \tl_remove_all:Nn #1 {~}
}
\cs_generate_variant:Nn \tl_replace_all:Nnn {Nnx}
```

Features

```
\@@_keys_define_code:nnn {fontspec-preparse} {UprightFeatures}
{
 \clist_set:Nn \l_@@_fontfeat_up_clist {#1}
}
\@@_keys_define_code:nnn {fontspec-preparse} {BoldFeatures}
{
 \clist_set:Nn \l_@@_fontfeat_bf_clist {#1}

%  \prop_put:NxV \l_@@_nfss_prop
%     {BoldFont-\g_@@_curr_series_tl} \l_@@_curr_bfname_tl
}
\@@_keys_define_code:nnn {fontspec-preparse} {ItalicFeatures}
{
 \clist_set:Nn \l_@@_fontfeat_it_clist {#1}
}
\@@_keys_define_code:nnn {fontspec-preparse} {BoldItalicFeatures}
{
 \clist_set:Nn \l_@@_fontfeat_bfit_clist {#1}
}
\@@_keys_define_code:nnn {fontspec-preparse} {SlantedFeatures}
{
 \clist_set:Nn \l_@@_fontfeat_sl_clist {#1}
```

```
 }
\@@_keys_define_code:nnn {fontspec-preparse} {BoldSlantedFeatures}
 {
  \clist_set:Nn \l_@@_fontfeat_bfsl_clist {#1}
 }
```

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

```
\@@_keys_define_code:nnn {fontspec} {SmallCapsFeatures}
 {
  \bool_if:NF \l_@@_firsttime_bool
   {
    \clist_set:Nn \l_@@_fontfeat_sc_clist {#1}
   }
 }
```

paragraphFeatures varying by size

```
\@@_keys_define_code:nnn {fontspec-preparse} {SizeFeatures}
 {
  \clist_set:Nn \l_@@_sizefeat_clist {#1}
  \clist_put_right:Nn \l_@@_fontfeat_up_clist { SizeFeatures = {#1} }
 }
\@@_keys_define_code:nnn {fontspec-preparse-nested} {SizeFeatures}
 {
  \clist_set:Nn \l_@@_sizefeat_clist {#1}
  \tl_if_empty:NT \l_@@_this_font_tl
   { \tl_set:Nn \l_@@_this_font_tl { -- } } % needs to be non-empty as a flag
 }
\@@_keys_define_code:nnn {fontspec-preparse-nested} {Font}
 {
  \tl_set:Nn \l_@@_this_font_tl {#1}
 }
\@@_keys_define_code:nnn {fontspec} {SizeFeatures}
 {
  % dummy
 }
\@@_keys_define_code:nnn {fontspec} {Font}
 {
  % dummy
 }
\@@_keys_define_code:nnn {fontspec-sizing} {Size}
 {
  \tl_set:Nn \l_@@_size_tl {#1}
 }
\@@_keys_define_code:nnn {fontspec-sizing} {Font}
 {
  \fontspec_complete_fontname:Nn \l_@@_sizedfont_tl {#1}
 }
```

23.6.4 Font-independent features

These features can be applied to any font.

NFSS family Interactions with other packages will sometimes require setting the NFSS family explicitly. (By default fontspec auto-generates one based on the font name.)

```
\@@_keys_define_code:nnn {fontspec-preparse} {NFSSFamily}
 {
  \tl_set:Nx \l_@@_nfss_fam_tl { #1 }
  \cs_undefine:c {g_@@_UID_\l_@@_fontid_tl}
  \tl_if_exist:NT \l_fontspec_family_tl
   { \cs_undefine:c {g_@@_ \l_fontspec_family_tl _prop} }
 }
```

NFSS series/shape This option looks similar in name but has a very different function.

```
\cs_generate_variant:Nn \prop_put:Nnn {Nxx}
\prop_new:N \l_@@_nfssfont_prop
\@@_keys_define_code:nnn {fontspec} {FontFace}
 {
  \tl_set:No \l_@@_arg_tl { \use_iii:nnn #1 }
  \tl_set_eq:NN \l_@@_this_feat_tl \l_@@_arg_tl
  \tl_clear:N \l_@@_this_font_tl
  \int_compare:nT { \clist_count:N \l_@@_arg_tl = 1 }
   {
⟨*debug⟩
    \typeout{FontFace~ parsing:~ one~ clist~ item}
⟨/debug⟩
    \tl_if_in:NnF \l_@@_arg_tl {=}
     {
⟨*debug⟩
      \typeout{FontFace~ parsing:~ no~ equals~ =>~ font~ name~ only}
⟨/debug⟩
      \tl_set_eq:NN \l_@@_this_font_tl \l_@@_arg_tl
      \tl_clear:N \l_@@_this_feat_tl
     }
   }

  \@@_add_nfssfont:oooo
   {\use_i:nnn #1}{\use_ii:nnn #1}{\l_@@_this_font_tl}{\l_@@_this_feat_tl}
 }
```

`\@@_add_nfssfont:nnnn` #1 : series
#2 : shape
#3 : fontname
#4 : fontspec features

```
\cs_new:Nn \@@_add_nfssfont:nnnn
 {
  \tl_set:Nx \l_@@_this_font_tl {#3}

  \tl_if_empty:xTF {#4}
   { \clist_set:Nn \l_@@_sizefeat_clist {Size={-}} }
   { \keys_set_known:noN {fontspec-preparse-nested} {#4} \l_@@_tmp_tl }

  \tl_if_empty:NF \l_@@_this_font_tl
   {
```

```
    \prop_put:Nxx \l_@@_nfssfont_prop {#1/#2}
     { {#1}{#2}{\l_@@_this_font_tl}{#4}{\l_@@_sizefeat_clist} }
   }
 }
\cs_generate_variant:Nn \@@_add_nfssfont:nnnn {ooo}
\cs_generate_variant:Nn \@@_add_nfssfont:nnnn {oooo}
\cs_generate_variant:Nn \tl_if_empty:nTF {x}
```

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.

```
\@@_keys_define_code:nnn {fontspec} {Scale}
{
 \str_case:nnn {#1}
  {
   {MatchLowercase} { \@@_calc_scale:n {5} }
   {MatchUppercase} { \@@_calc_scale:n {8} }
  }
  { \tl_set:Nx \l_@@_scale_tl {#1} }
 \tl_set:Nx \l_@@_scale_tl { s*[\l_@@_scale_tl] }
}
```

`\@@_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:Nn \@@_calc_scale:n
{
 \group_begin:
  \rmfamily
  \@@_set_font_dimen:NnN \l_@@_tmpa_dim {#1} \font
  \@@_set_font_dimen:NnN \l_@@_tmpb_dim {#1} \l_fontspec_font
  \tl_gset:Nx \l_@@_scale_tl
   {
    \fp_eval:n { \dim_to_fp:n {\l_@@_tmpa_dim} /
                 \dim_to_fp:n {\l_@@_tmpb_dim} }
   }
  \@@_info:n {set-scale}
 \group_end:
}
```

`\@@_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:Nn \@@_set_font_dimen:NnN
{
 \dim_set:Nn #1 { \fontdimen #2 #3 }
 \dim_compare:nNnT #1 = {0pt}
```

```
  {
   \settoheight #1
    {
     \str_if_eq:nnTF {#3} {\font} \rmfamily #3
     \int_case: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.

```
\@@_keys_define_code:nnn {fontspec} {WordSpace}
{
 \bool_if:NF \l_@@_firsttime_bool
  { \_fontspec_parse_wordspace:w #1,,,\q_stop }
}
```

`\_fontspec_parse_wordspace:w` 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}`.

```
\cs_set:Npn \_fontspec_parse_wordspace:w #1,#2,#3,#4 \q_stop
{
 \tl_if_empty:nTF {#4}
  {
   \tl_set:Nn \l_@@_wordspace_adjust_tl
    {
     \fontdimen 2 \font = #1 \fontdimen 2 \font
     \fontdimen 3 \font = #1 \fontdimen 3 \font
     \fontdimen 4 \font = #1 \fontdimen 4 \font
    }
  }
  {
   \tl_set:Nn \l_@@_wordspace_adjust_tl
    {
     \fontdimen 2 \font = #1 \fontdimen 2 \font
     \fontdimen 3 \font = #2 \fontdimen 3 \font
     \fontdimen 4 \font = #3 \fontdimen 4 \font
    }
  }
}
```

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

```
\@@_keys_define_code:nnn {fontspec} {PunctuationSpace}
{
 \str_case_x:nnn {#1}
  {
   {WordSpace}
   {
    \tl_set:Nn \l_@@_punctspace_adjust_tl
```

```
      { \fontdimen 7 \font = 0 \fontdimen 2 \font }
    }
    {TwiceWordSpace}
    {
     \tl_set:Nn \l_@@_punctspace_adjust_tl
      { \fontdimen 7 \font = 1 \fontdimen 2 \font }
    }
   }
   {
     \tl_set:Nn \l_@@_punctspace_adjust_tl
     { \fontdimen 7 \font = #1 \fontdimen 7 \font }
   }
}
```

Secret hook into the font-adjustment code

```
\@@_keys_define_code:nnn {fontspec} {FontAdjustment}
{
 \tl_put_right:Nx \l_@@_postadjust_tl {#1}
}
```

Letterspacing

```
\@@_keys_define_code:nnn {fontspec} {LetterSpace}
{
 \@@_update_featstr:n {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.

```
\@@_keys_define_code:nnn {fontspec} {HyphenChar}
{
 \str_if_eq:nnTF {#1} {None}
  {
   \tl_put_right:Nn \l_@@_postadjust_tl
     { \hyphenchar \font = \c_minus_one }
  }
  {
   \tl_if_single:nTF {#1}
    { \tl_set:Nn \l_fontspec_hyphenchar_tl {`#1} }
    { \tl_set:Nn \l_fontspec_hyphenchar_tl { #1} }
   \font_glyph_if_exist:NnTF \l_fontspec_font {\l_fontspec_hyphenchar_tl}
    {
     \tl_put_right:Nn \l_@@_postadjust_tl
⟨*xetexx⟩
       { \hyphenchar \font = \l_fontspec_hyphenchar_tl \scan_stop: }
⟨/xetexx⟩
⟨*luatex⟩
       {
         \hyphenchar \font = \c_zero
         \luatexprehyphenchar = \l_fontspec_hyphenchar_tl \scan_stop:
```

```
        }
⟨/luatex⟩
    }
    { \@@_error:nx {no-glyph}{#1} }
  }
}
```

Color Hooks into pkgxcolor, which names its colours `\color@<name>`.

```
\@@_keys_define_code:nnn {fontspec} {Color}
{
 \cs_if_exist:cTF { \token_to_str:N \color@ #1 }
  {
   \convertcolorspec{named}{#1}{HTML}\l_@@_hexcol_tl
  }
  {
   \int_compare:nTF { \tl_count:n {#1} == 6 }
    { \tl_set:Nn \l_@@_hexcol_tl {#1} }
    {
     \int_compare:nTF { \tl_count:n {#1} == 8 }
      { \fontspec_parse_colour:viii #1 }
      {
       \bool_if:NF \l_@@_firsttime_bool
        { \@@_warning:nx {bad-colour} {#1} }
      }
    }
  }
}
\cs_set:Npn \fontspec_parse_colour:viii #1#2#3#4#5#6#7#8
{
 \tl_set:Nn \l_@@_hexcol_tl {#1#2#3#4#5#6}
 \tl_if_eq:NNF \l_@@_opacity_tl \g_@@_opacity_tl
  {
   \bool_if:NF \l_@@_firsttime_bool
    { \@@_warning:nx {opa-twice-col} {#7#8} }
  }
 \tl_set:Nn \l_@@_opacity_tl {#7#8}
}
\aliasfontfeature{Color}{Colour}
\int_new:N \l_@@_tmp_int
\@@_keys_define_code:nnn {fontspec} {Opacity}
{
 \int_set:Nn \l_@@_tmp_int {255}
 \@@_int_mult_truncate:Nn \l_@@_tmp_int { #1 }
 \tl_if_eq:NNF \l_@@_opacity_tl \g_@@_opacity_tl
  {
   \bool_if:NF \l_@@_firsttime_bool
    { \@@_warning:nx {opa-twice} {#1} }
  }
 \tl_set:Nx \l_@@_opacity_tl
  {
    \int_compare:nT { \l_@@_tmp_int <= "F } {0} % zero pad
```

```
    \int_to_hexadecimal:n { \l_@@_tmp_int }
   }
}
```

Mapping

```
\@@_keys_define_code:nnn {fontspec} {Mapping}
⟨*xetexx⟩
{
 \@@_update_featstr:n { mapping = #1 }
}
⟨/xetexx⟩
⟨*luatex⟩
{
 \str_if_eq:nnTF {#1} {tex-text}
  {
   \@@_warning:n {no-mapping-ligtex}
   \msg_redirect_name:nnn {fontspec} {no-mapping-ligtex} {none}
   \keys_set:nn {fontspec} { Ligatures=TeX }
  }
  { \@@_warning:n {no-mapping} }
}
⟨/luatex⟩
```

FeatureFile

```
\@@_keys_define_code:nnn {fontspec} {FeatureFile}
{
 \@@_update_featstr:n { featurefile = #1 }
}
```

23.6.5 Continuous font axes

```
\@@_keys_define_code:nnn {fontspec} {Weight}
{
 \@@_update_featstr:n{weight=#1}
}
\@@_keys_define_code:nnn {fontspec} {Width}
{
 \@@_update_featstr:n{width=#1}
}
\@@_keys_define_code:nnn {fontspec} {OpticalSize}
⟨*xetexx⟩
{
 \bool_if:NTF \l_@@_ot_bool
  {
   \tl_set:Nn \l_@@_optical_size_tl {/ S = #1}
  }
  {
   \bool_if:NT \l_@@_mm_bool
    {
     \@@_update_featstr:n { optical size = #1 }
    }
```

```
    }
   \bool_if:nT { !\l_@@_ot_bool && !\l_@@_mm_bool }
    {
     \bool_if:NT \l_@@_firsttime_bool
      { \@@_warning:n {no-opticals} }
    }
  }
⟨/xetexx⟩
⟨*luatex⟩
 {
  \tl_set:Nn \l_@@_optical_size_tl {/ S = #1}
 }
⟨/luatex⟩
```

23.6.6 Font transformations

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

```
\keys_define:nn {fontspec}
 {
  FakeSlant .code:n =
   {
    \@@_update_featstr:n{slant=#1}
   },
  FakeSlant .default:n = {0.2}
}
\keys_define:nn {fontspec}
 {
  FakeStretch .code:n =
   {
    \@@_update_featstr:n{extend=#1}
   },
  FakeStretch .default:n = {1.2}
}
⟨*xetexx⟩
\keys_define:nn {fontspec}
 {
  FakeBold .code:n =
   {
    \@@_update_featstr:n {embolden=#1}
   },
  FakeBold .default:n = {1.5}
 }
⟨/xetexx⟩
⟨*luatex⟩
\keys_define:nn {fontspec}
 {
  FakeBold .code:n = { \@@_warning:n {fakebold-only-xetex} }
 }
⟨/luatex⟩
```

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.)

```
\keys_define:nn {fontspec}
 {
  AutoFakeSlant .code:n =
   {
    \bool_if:NT \l_@@_firsttime_bool
     {
      \tl_set:Nn \l_fontspec_fake_slant_tl {#1}
      \clist_put_right:Nn \l_@@_fontfeat_it_clist {FakeSlant=#1}
      \tl_set_eq:NN \l_fontspec_fontname_it_tl \l_fontspec_fontname_tl
      \bool_set_false:N \l_@@_noit_bool

      \tl_if_empty:NF \l_fontspec_fake_embolden_tl
       {
        \clist_put_right:Nx \l_@@_fontfeat_bfit_clist
         {FakeBold=\l_fontspec_fake_embolden_tl}
        \clist_put_right:Nx \l_@@_fontfeat_bfit_clist {FakeSlant=#1}
        \tl_set_eq:NN \l_fontspec_fontname_bfit_tl \l_fontspec_fontname_tl
       }
     }
   },
  AutoFakeSlant .default:n = {0.2}
}
```

Same but reversed:

```
\keys_define:nn {fontspec}
 {
  AutoFakeBold .code:n =
   {
    \bool_if:NT \l_@@_firsttime_bool
     {
      \tl_set:Nn \l_fontspec_fake_embolden_tl {#1}
      \clist_put_right:Nn \l_@@_fontfeat_bf_clist {FakeBold=#1}
      \tl_set_eq:NN \l_fontspec_fontname_bf_tl \l_fontspec_fontname_tl
      \bool_set_false:N \l_@@_nobf_bool

      \tl_if_empty:NF \l_fontspec_fake_slant_tl
       {
        \clist_put_right:Nx \l_@@_fontfeat_bfit_clist
         {FakeSlant=\l_fontspec_fake_slant_tl}
        \clist_put_right:Nx \l_@@_fontfeat_bfit_clist {FakeBold=#1}
        \tl_set_eq:NN \l_fontspec_fontname_bfit_tl \l_fontspec_fontname_tl
       }
     }
   },
  AutoFakeBold .default:n = {1.5}
}
```

23.6.7 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 [...]). Both AAT and OpenType names are offered to chose `Rare/Discretionary` ligatures.

```
\@@_define_font_feature:n{Ligatures}
\@@_define_feature_option:nnnnn{Ligatures}{Required}       {1}{0}{+rlig}
\@@_define_feature_option:nnnnn{Ligatures}{NoRequired}     {1}{1}{-rlig}
\@@_define_feature_option:nnnnn{Ligatures}{Common}         {1}{2}{+liga}
\@@_define_feature_option:nnnnn{Ligatures}{NoCommon}       {1}{3}{-liga}
\@@_define_feature_option:nnnnn{Ligatures}{Rare}           {1}{4}{+dlig}
\@@_define_feature_option:nnnnn{Ligatures}{NoRare}         {1}{5}{-dlig}
\@@_define_feature_option:nnnnn{Ligatures}{Discretionary}  {1}{4}{+dlig}
\@@_define_feature_option:nnnnn{Ligatures}{NoDiscretionary}{1}{5}{-dlig}
\@@_define_feature_option:nnnnn{Ligatures}{Contextual}     {}{}  {+clig}
\@@_define_feature_option:nnnnn{Ligatures}{NoContextual}   {}{}  {-clig}
\@@_define_feature_option:nnnnn{Ligatures}{Historic}       {}{}  {+hlig}
\@@_define_feature_option:nnnnn{Ligatures}{NoHistoric}     {}{}  {-hlig}
\@@_define_feature_option:nnnnn{Ligatures}{Logos}          {1}{6} {}
\@@_define_feature_option:nnnnn{Ligatures}{NoLogos}        {1}{7} {}
\@@_define_feature_option:nnnnn{Ligatures}{Rebus}          {1}{8} {}
\@@_define_feature_option:nnnnn{Ligatures}{NoRebus}        {1}{9} {}
\@@_define_feature_option:nnnnn{Ligatures}{Diphthong}      {1}{10}{}
\@@_define_feature_option:nnnnn{Ligatures}{NoDiphthong}    {1}{11}{}
\@@_define_feature_option:nnnnn{Ligatures}{Squared}        {1}{12}{}
\@@_define_feature_option:nnnnn{Ligatures}{NoSquared}      {1}{13}{}
\@@_define_feature_option:nnnnn{Ligatures}{AbbrevSquared}  {1}{14}{}
\@@_define_feature_option:nnnnn{Ligatures}{NoAbbrevSquared}{1}{15}{}
\@@_define_feature_option:nnnnn{Ligatures}{Icelandic}      {1}{32}{}
\@@_define_feature_option:nnnnn{Ligatures}{NoIcelandic}    {1}{33}{}
```

Emulate CM extra ligatures.

```
\keys_define:nn {fontspec}
 {
  Ligatures / TeX .code:n =
   {
⟨*xetexx⟩
    \@@_update_featstr:n { mapping = tex-text }
⟨/xetexx⟩
⟨*luatex⟩
    \@@_update_featstr:n { +tlig; +trep }
⟨/luatex⟩
   }
 }
```

23.6.8 Letters

```
\@@_define_font_feature:n{Letters}
\@@_define_feature_option:nnnnn{Letters}{Normal}            {3}{0}{}
\@@_define_feature_option:nnnnn{Letters}{Uppercase}         {3}{1}{+case}
\@@_define_feature_option:nnnnn{Letters}{Lowercase}         {3}{2}{}
\@@_define_feature_option:nnnnn{Letters}{SmallCaps}         {3}{3}{+smcp}
\@@_define_feature_option:nnnnn{Letters}{PetiteCaps}        {} {} {+pcap}
```

```
\@@_define_feature_option:nnnnn{Letters}{UppercaseSmallCaps} {} {} {+c2sc}
\@@_define_feature_option:nnnnn{Letters}{UppercasePetiteCaps}{} {} {+c2pc}
\@@_define_feature_option:nnnnn{Letters}{InitialCaps}        {3}{4}{}
\@@_define_feature_option:nnnnn{Letters}{Unicase}            {} {} {+unic}
\@@_define_feature_option:nnnnn{Letters}{Random}             {} {} {+rand}
```

23.6.9 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.

```
\@@_define_font_feature:n{Numbers}
\@@_define_feature_option:nnnnn{Numbers}{Monospaced}   {6} {0}{+tnum}
\@@_define_feature_option:nnnnn{Numbers}{Proportional} {6} {1}{+pnum}
\@@_define_feature_option:nnnnn{Numbers}{Lowercase}    {21}{0}{+onum}
\@@_define_feature_option:nnnnn{Numbers}{OldStyle}     {21}{0}{+onum}
\@@_define_feature_option:nnnnn{Numbers}{Uppercase}    {21}{1}{+lnum}
\@@_define_feature_option:nnnnn{Numbers}{Lining}       {21}{1}{+lnum}
\@@_define_feature_option:nnnnn{Numbers}{SlashedZero}  {14}{5}{+zero}
\@@_define_feature_option:nnnnn{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
{
 \@@_define_feature_option:nnnnn{Numbers}{Arabic}{}{}{+anum}
}
```

23.6.10 Contextuals

```
\@@_define_font_feature:n  {Contextuals}
\@@_define_feature_option:nnnnn{Contextuals}{Swash}        {} {} {+cswh}
\@@_define_feature_option:nnnnn{Contextuals}{NoSwash}      {} {} {-cswh}
\@@_define_feature_option:nnnnn{Contextuals}{Alternate}    {} {} {+calt}
\@@_define_feature_option:nnnnn{Contextuals}{NoAlternate}  {} {} {-calt}
\@@_define_feature_option:nnnnn{Contextuals}{WordInitial}  {8}{0}{+init}
\@@_define_feature_option:nnnnn{Contextuals}{NoWordInitial}{8}{1}{-init}
\@@_define_feature_option:nnnnn{Contextuals}{WordFinal}    {8}{2}{+fina}
\@@_define_feature_option:nnnnn{Contextuals}{NoWordFinal}  {8}{3}{-fina}
\@@_define_feature_option:nnnnn{Contextuals}{LineInitial}  {8}{4}{}
\@@_define_feature_option:nnnnn{Contextuals}{NoLineInitial}{8}{5}{}
\@@_define_feature_option:nnnnn{Contextuals}{LineFinal}    {8}{6}{+falt}
\@@_define_feature_option:nnnnn{Contextuals}{NoLineFinal}  {8}{7}{-falt}
\@@_define_feature_option:nnnnn{Contextuals}{Inner}        {8}{8}{+medi}
\@@_define_feature_option:nnnnn{Contextuals}{NoInner}      {8}{9}{-medi}
```

23.6.11 Diacritics

```
\@@_define_font_feature:n{Diacritics}
\@@_define_feature_option:nnnnn{Diacritics}{Show}        {9}{0}{}
\@@_define_feature_option:nnnnn{Diacritics}{Hide}        {9}{1}{}
\@@_define_feature_option:nnnnn{Diacritics}{Decompose}   {9}{2}{}
\@@_define_feature_option:nnnnn{Diacritics}{MarkToBase}  {}{}{+mark}
```

```
\@@_define_feature_option:nnnnn{Diacritics}{NoMarkToBase}{}{}{-mark}
\@@_define_feature_option:nnnnn{Diacritics}{MarkToMark}  {}{}{+mkmk}
\@@_define_feature_option:nnnnn{Diacritics}{NoMarkToMark}{}{}{-mkmk}
\@@_define_feature_option:nnnnn{Diacritics}{AboveBase}   {}{}{+abvm}
\@@_define_feature_option:nnnnn{Diacritics}{NoAboveBase} {}{}{-abvm}
\@@_define_feature_option:nnnnn{Diacritics}{BelowBase}   {}{}{+blwm}
\@@_define_feature_option:nnnnn{Diacritics}{NoBelowBase} {}{}{-blwm}
```

23.6.12 Kerning

```
\@@_define_font_feature:n{Kerning}
\@@_define_feature_option:nnnnn{Kerning}{Uppercase}{}{}{+cpsp}
\@@_define_feature_option:nnnnn{Kerning}{On}       {}{}{+kern}
\@@_define_feature_option:nnnnn{Kerning}{Off}      {}{}{-kern}
%\@@_define_feature_option:nnnnn{Kerning}{Vertical}{}{}{+vkrn}
%\@@_define_feature_option:nnnnn{Kerning}
%    {VerticalAlternateProportional}{}{}{+vpal}
%\@@_define_feature_option:nnnnn{Kerning}{VerticalAlternateHalfWidth}{}{}{+vhal}
```

23.6.13 Vertical position

```
\@@_define_font_feature:n{VerticalPosition}
\@@_define_feature_option:nnnnn{VerticalPosition}{Normal}     {10}{0}{}
\@@_define_feature_option:nnnnn{VerticalPosition}{Superior}   {10}{1}{+sups}
\@@_define_feature_option:nnnnn{VerticalPosition}{Inferior}   {10}{2}{+subs}
\@@_define_feature_option:nnnnn{VerticalPosition}{Ordinal}    {10}{3}{+ordn}
\@@_define_feature_option:nnnnn{VerticalPosition}{Numerator}  {}  {} {+numr}
\@@_define_feature_option:nnnnn{VerticalPosition}{Denominator}{}  {} {+dnom}
\@@_define_feature_option:nnnnn{VerticalPosition}{ScientificInferior}{}{}{+sinf}
```

23.6.14 Fractions

```
\@@_define_font_feature:n{Fractions}
\@@_define_feature_option:nnnnn{Fractions}{On}       {11}{1}{+frac}
\@@_define_feature_option:nnnnn{Fractions}{Off}      {11}{0}{-frac}
\@@_define_feature_option:nnnnn{Fractions}{Diagonal} {11}{2}{}
\@@_define_feature_option:nnnnn{Fractions}{Alternate}{}  {} {+afrc}
```

23.6.15 Alternates and variants

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

```
\@@_define_font_feature:n { Alternate }
\keys_define:nn {fontspec}
 {
  Alternate .default:n = {0} ,
  Alternate / unknown .code:n =
   {
    \clist_map_inline:nn {#1}
      { \fontspec_make_feature:nnx {17}{##1} { \fontspec_salt:n {##1} } }
   }
 }
\cs_set:Nn \fontspec_salt:n { +salt = #1 }
\@@_define_font_feature:n {Variant}
```

```
\keys_define:nn {fontspec}
 {
  Variant .default:n = {0} ,
  Variant / unknown .code:n =
   {
    \clist_map_inline:nn {#1}
      { \fontspec_make_feature:nnx {18}{##1} { +ss \two@digits {##1} } }
   }
 }
\aliasfontfeature{Variant}{StylisticSet}
\@@_define_font_feature:n { CharacterVariant }
\use:x
 {
  \cs_new:Npn \exp_not:N \fontspec_parse_cv:w
      ##1 \c_colon_str ##2 \c_colon_str ##3 \exp_not:N \q_nil
   {
     \fontspec_make_numbered_feature:xn
       { +cv \exp_not:N \two@digits {##1} } {##2}
   }
  \keys_define:nn {fontspec}
   {
    CharacterVariant / unknown .code:n =
     {
      \clist_map_inline:nn {##1}
       {
        \exp_not:N \fontspec_parse_cv:w
          ####1 \c_colon_str 0 \c_colon_str \exp_not:N \q_nil
       }
     }
   }
 }
```

Possibilities: `a:0:\q_nil` or `a:b:0:\q_nil`.

23.6.16 Style

```
\@@_define_font_feature:n{Style}
\@@_define_feature_option:nnnnn{Style}{Alternate}     {}  {} {+salt}
\@@_define_feature_option:nnnnn{Style}{Italic}        {32}{2}{+ital}
\@@_define_feature_option:nnnnn{Style}{Ruby}          {28}{2}{+ruby}
\@@_define_feature_option:nnnnn{Style}{Swash}         {}  {} {+swsh}
\@@_define_feature_option:nnnnn{Style}{Historic}      {}  {} {+hist}
\@@_define_feature_option:nnnnn{Style}{Display}       {19}{1}{}
\@@_define_feature_option:nnnnn{Style}{Engraved}      {19}{2}{}
\@@_define_feature_option:nnnnn{Style}{TitlingCaps}   {19}{4}{+titl}
\@@_define_feature_option:nnnnn{Style}{TallCaps}      {19}{5}{}
\@@_define_feature_option:nnnnn{Style}{HorizontalKana}{}  {} {+hkna}
\@@_define_feature_option:nnnnn{Style}{VerticalKana}  {}  {} {+vkna}
\fontspec_define_numbered_feat:nnnn {Style} {MathScript}       {+ssty} {0}
\fontspec_define_numbered_feat:nnnn {Style} {MathScriptScript} {+ssty} {1}
```

23.6.17 CJK shape

```
\@@_define_font_feature:n{CJKShape}
\@@_define_feature_option:nnnnn{CJKShape}{Traditional}{20}{0} {+trad}
\@@_define_feature_option:nnnnn{CJKShape}{Simplified} {20}{1} {+smpl}
\@@_define_feature_option:nnnnn{CJKShape}{JIS1978}    {20}{2} {+jp78}
\@@_define_feature_option:nnnnn{CJKShape}{JIS1983}    {20}{3} {+jp83}
\@@_define_feature_option:nnnnn{CJKShape}{JIS1990}    {20}{4} {+jp90}
\@@_define_feature_option:nnnnn{CJKShape}{Expert}     {20}{10}{+expt}
\@@_define_feature_option:nnnnn{CJKShape}{NLC}        {20}{13}{+nlck}
```

23.6.18 Character width

```
\@@_define_font_feature:n{CharacterWidth}
\@@_define_feature_option:nnnnn{CharacterWidth}{Proportional}{22}{0}{+pwid}
\@@_define_feature_option:nnnnn{CharacterWidth}{Full}{22}{1}{+fwid}
\@@_define_feature_option:nnnnn{CharacterWidth}{Half}{22}{2}{+hwid}
\@@_define_feature_option:nnnnn{CharacterWidth}{Third}{22}{3}{+twid}
\@@_define_feature_option:nnnnn{CharacterWidth}{Quarter}{22}{4}{+qwid}
\@@_define_feature_option:nnnnn{CharacterWidth}{AlternateProportional}{22}{5}{+palt}
\@@_define_feature_option:nnnnn{CharacterWidth}{AlternateHalf}{22}{6}{+halt}
\@@_define_feature_option:nnnnn{CharacterWidth}{Default}{22}{7}{}
```

23.6.19 Annotation

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

```
\@@_define_font_feature:n { Annotation }
\keys_define:nn {fontspec}
 {
  Annotation .default:n = {0} ,
  Annotation / unknown .code:n =
   {
    \fontspec_make_feature:nnx {}{}{ +nalt=#1 }
   }
 }
```

23.6.20 Vertical

```
\keys_define:nn {fontspec}
 {
  Vertical .choice: ,
  Vertical / RotatedGlyphs .code:n =
   {
    \bool_if:NTF \l_@@_ot_bool
     {
```

```
      \fontspec_make_feature:nnn{}{}{+vrt2}
      \@@_update_featstr:n {vertical}
    }
    {
     \@@_update_featstr:n {vertical}
    }
  }
}
```

23.6.21 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}
```

23.6.22 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{Parsi}{FAR}
\newfontlanguage{Persian}{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:

```
\keys_define:nn {fontspec}
 {
  Language / Turkish .code:n =
   {
```

```
    \fontspec_check_lang:nTF {TRK}
     {
      \int_set:Nn \l_fontspec_language_int {\l_fontspec_strnum_int}
      \tl_set:Nn \l_fontspec_lang_tl {TRK}
     }
     {
      \fontspec_check_lang:nTF {TUR}
       {
        \int_set:Nn \l_fontspec_language_int {\l_fontspec_strnum_int}
        \tl_set:Nn \l_fontspec_lang_tl {TUR}
       }
       {
        \@@_warning:nx {language-not-exist} {Turkish}
        \keys_set:nn {fontspec} {Language=Default}
       }
     }
   }
}
```

Default

```
\@@_keys_define_code:nnn {fontspec}{ Language / Default }
{
 \tl_set:Nn \l_fontspec_lang_tl {DFLT}
 \int_zero:N \l_fontspec_language_int
}
```

23.6.23 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.

```
\@@_keys_define_code:nnn {fontspec} {RawFeature}
{
 \@@_update_featstr:n {#1}
}
```

23.7 Italic small caps

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

`\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}
```

\fontspec_blend_shape:nnn 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.

```
\cs_new:Nn \fontspec_blend_shape:nnn
{
 \bool_if:nTF
  {
   \str_if_eq_x_p:nn {\f@shape} {#2} &&
   \cs_if_exist_p:c {\f@encoding/\f@family/\f@series/#3}
  }
  { \fontshape{#3}\selectfont }
  { \fontshape{#1}\selectfont }
}
```

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

```
\DeclareRobustCommand \itshape
{
 \not@math@alphabet\itshape\mathit
 \fontspec_blend_shape:nnn\itdefault\scdefault\sidefault
}
\DeclareRobustCommand \slshape
{
 \not@math@alphabet\slshape\relax
 \fontspec_blend_shape:nnn\sldefault\scdefault\sidefault
}
\DeclareRobustCommand \scshape
{
 \not@math@alphabet\scshape\relax
 \fontspec_blend_shape:nnn\scdefault\itdefault\sidefault
}
\DeclareRobustCommand \upshape
{
 \not@math@alphabet\upshape\relax
 \fontspec_blend_shape:nnn\updefault\sidefault\scdefault
}
```

23.8 Selecting maths fonts

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

\fontspec_setup_maths: 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}
{
 \bool_set_true:N \g_@@_pkg_euler_loaded_bool
}
```

```
{
 \bool_set_false:N \g_@@_pkg_euler_loaded_bool
}
\cs_set:Nn \fontspec_setup_maths:
{
 \@ifpackageloaded{euler}
  {
   \bool_if:NTF \g_@@_pkg_euler_loaded_bool
    { \bool_set_true:N \g_@@_math_euler_bool }
    { \@@_error:n {euler-too-late} }
  }
  {}
 \@ifpackageloaded{lucbmath}{\bool_set_true:N \g_@@_math_lucida_bool}{}
 \@ifpackageloaded{lucidabr}{\bool_set_true:N \g_@@_math_lucida_bool}{}
 \@ifpackageloaded{lucimatx}{\bool_set_true:N \g_@@_math_lucida_bool}{}
```

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.

```
\group_begin:
  \mathchardef\@tempa="603A \relax
  \ifx\colon\@tempa
    \DeclareMathSymbol{\colon}{\mathpunct}{legacymaths}{58}
  \fi
\group_end:
```

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

```
\bool_if:NF \g_@@_math_euler_bool
 {
  \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.

```
  \bool_if:NF \g_@@_math_lucida_bool
   {
    \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}
    }
  }
```

Finally, we change the font definitions for `\mathrm` and so on. These are defined using the `\g_@@_mathrm_tl` (…) 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 `\g_@@_bfmathrm_tl`.

```
 \DeclareSymbolFont{operators}\g_fontspec_encoding_tl\g_@@_mathrm_tl\mddefault\updefault
 \SetSymbolFont{operators}{normal}\g_fontspec_encoding_tl\g_@@_mathrm_tl\mddefault\updefault
  \DeclareSymbolFontAlphabet\mathrm{operators}
 \SetMathAlphabet\mathit{normal}\g_fontspec_encoding_tl\g_@@_mathrm_tl\mddefault\itdefault
 \SetMathAlphabet\mathbf{normal}\g_fontspec_encoding_tl\g_@@_mathrm_tl\bfdefault\updefault
 \SetMathAlphabet\mathsf{normal}\g_fontspec_encoding_tl\g_@@_mathsf_tl\mddefault\updefault
 \SetMathAlphabet\mathtt{normal}\g_fontspec_encoding_tl\g_@@_mathtt_tl\mddefault\updefault
 \SetSymbolFont{operators}{bold}\g_fontspec_encoding_tl\g_@@_mathrm_tl\bfdefault\updefault
  \tl_if_empty:NTF \g_@@_bfmathrm_tl
   {
    \SetMathAlphabet\mathit{bold}\g_fontspec_encoding_tl\g_@@_mathrm_tl\bfdefault\itdefault
   }
   {
    \SetMathAlphabet\mathrm{bold}\g_fontspec_encoding_tl\g_@@_bfmathrm_tl\mddefault\updefault
    \SetMathAlphabet\mathbf{bold}\g_fontspec_encoding_tl\g_@@_bfmathrm_tl\bfdefault\updefault
    \SetMathAlphabet\mathit{bold}\g_fontspec_encoding_tl\g_@@_bfmathrm_tl\mddefault\itdefault
   }
 \SetMathAlphabet\mathsf{bold}\g_fontspec_encoding_tl\g_@@_mathsf_tl\bfdefault\updefault
 \SetMathAlphabet\mathtt{bold}\g_fontspec_encoding_tl\g_@@_mathtt_tl\bfdefault\updefault
}
```

`\fontspec_maybe_setup_maths:`

We're a little less sophisticated about not executing the maths setup 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\bool_set_false:N \g_@@_math_bool\fi` be a better test? This needs more cooperation with euler and lucida, I think.

```
\cs_new:Nn \fontspec_maybe_setup_maths:
{
 \@ifpackageloaded{anttor}
  {
   \ifx\define@antt@mathversions a\bool_set_false:N \g_@@_math_bool\fi
  }{}
 \@ifpackageloaded{arev}{\bool_set_false:N \g_@@_math_bool}{}
 \@ifpackageloaded{eulervm}{\bool_set_false:N \g_@@_math_bool}{}
 \@ifpackageloaded{mathdesign}{\bool_set_false:N \g_@@_math_bool}{}
 \@ifpackageloaded{concmath}{\bool_set_false:N \g_@@_math_bool}{}
 \@ifpackageloaded{cmbright}{\bool_set_false:N \g_@@_math_bool}{}
 \@ifpackageloaded{mathesf}{\bool_set_false:N \g_@@_math_bool}{}
```

```
  \@ifpackageloaded{gfsartemisia}{\bool_set_false:N \g_@@_math_bool}{}
  \@ifpackageloaded{gfsneohellenic}{\bool_set_false:N \g_@@_math_bool}{}
  \@ifpackageloaded{iwona}
   {
    \ifx\define@iwona@mathversions a\bool_set_false:N \g_@@_math_bool\fi
   }{}
  \@ifpackageloaded{kpfonts}{\bool_set_false:N \g_@@_math_bool}{}
  \@ifpackageloaded{kmath}{\bool_set_false:N \g_@@_math_bool}{}
  \@ifpackageloaded{kurier}
   {
    \ifx\define@kurier@mathversions a\bool_set_false:N \g_@@_math_bool\fi
   }{}
  \@ifpackageloaded{fouriernc}{\bool_set_false:N \g_@@_math_bool}{}
  \@ifpackageloaded{fourier}{\bool_set_false:N \g_@@_math_bool}{}
  \@ifpackageloaded{lmodern}{\bool_set_false:N \g_@@_math_bool}{}
  \@ifpackageloaded{mathpazo}{\bool_set_false:N \g_@@_math_bool}{}
  \@ifpackageloaded{mathptmx}{\bool_set_false:N \g_@@_math_bool}{}
  \@ifpackageloaded{MinionPro}{\bool_set_false:N \g_@@_math_bool}{}
  \@ifpackageloaded{unicode-math}{\bool_set_false:N \g_@@_math_bool}{}
  \@ifpackageloaded{breqn}{\bool_set_false:N \g_@@_math_bool}{}
  \bool_if:NT \g_@@_math_bool
   {
    \@@_info:n {setup-math}
    \fontspec_setup_maths:
   }
 }
\AtBeginDocument{\fontspec_maybe_setup_maths:}
```

23.9 Finishing up

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

```
\bool_if:NT \g_@@_cfg_bool
 {
  \InputIfFileExists{fontspec.cfg}
    {}
    {\typeout{No~ fontspec.cfg~ file~ found;~ no~ configuration~ loaded.}}
 }
```

23.10 Compatibility

`\zf@enc` `\zf@family` `\zf@basefont` `\zf@fontspec`

Old interfaces. These are needed by, at least, the mathspec package.

```
\tl_set:Nn \zf@enc { \g_fontspec_encoding_tl }
\cs_set:Npn \zf@fontspec #1 #2
 {
  \fontspec_select:nn {#1} {#2}
  \tl_set:Nn \zf@family { \l_fontspec_family_tl }
  \tl_set:Nn \zf@basefont { \l_fontspec_font }
 }
```

The end! Thanks for coming.

```
\ExplSyntaxOff
⟨/fontspec & (xetexx | luatex)⟩
```

Part VIII
fontspec.lua

1 ⟨∗lua⟩

First we define some metadata.

```
2 fontspec          = fontspec or {}
3 local fontspec    = fontspec
4 fontspec.module   = {
5     name          = "fontspec",
6     version       = "2.3c",
7     date          = "2013/05/20",
8     description   = "Advanced font selection for LuaLaTeX.",
9     author        = "Khaled Hosny, Philipp Gesang",
10    copyright     = "Khaled Hosny, Philipp Gesang",
11    license       = "LPPL"
12 }
13
14 local err, warn, info, log = luatexbase.provides_module(fontspec.module)
15
```

Some utility functions

```
16 fontspec.log     = log
17 fontspec.warning = warn
18 fontspec.error   = err
19
20 function fontspec.sprint (...)
21     tex.sprint(luatexbase.catcodetables['latex-package'], ...)
22 end
```

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

```
23 local check_script   = luaotfload.aux.provides_script
24 local check_language = luaotfload.aux.provides_language
25 local check_feature  = luaotfload.aux.provides_feature
```

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

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

28 function fontspec.check_ot_script(fnt, script)
29     if check_script(font.id(fnt), script) then
30         tempswatrue()
31     else
32         tempswafalse()
33     end
34 end

35 function fontspec.check_ot_lang(fnt, lang, script)
36     if check_language(font.id(fnt), script, lang) then
37         tempswatrue()
38     else
39         tempswafalse()
40     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), script, lang, feat) then
        tempswatrue()
    else
        tempswafalse()
    end
end
local get_math_dimension = luaotfload.aux.get_math_dimension
function fontspec.mathfontdimen(fnt, str)
    local mathdimens = get_math_dimension(fnt, str)
    if mathdimens then
        fontspec.sprint(mathdimens)
        fontspec.sprint("sp")
    else
        fontspec.sprint("0pt")
    end
end
```

⟨/lua⟩

Part IX
fontspec-patches.sty

```
⟨*patches⟩
\ExplSyntaxOn
```

23.11 Unicode footnote symbols

This is handled by fixltx2e / LaTeX2015 now.

```
\cs_if_exist:NF \TextOrMath
{
 % copy official definition:
 \protected\expandafter\def\csname TextOrMath\space\endcsname{%
   \ifmmode  \expandafter\@secondoftwo
   \else     \expandafter\@firstoftwo  \fi}
 \edef\TextOrMath#1#2{%
   \expandafter\noexpand\csname TextOrMath\space\endcsname
   {#1}{#2}}
 % translation of official definition:
 \cs_set:Npn \@fnsymbol #1
  {
   \int_case:nnF {#1}
    {
     {0} {}
     {1} { \TextOrMath \textasteriskcentered* }
     {2} { \TextOrMath \textdagger\dagger }
     {3} { \TextOrMath \textdaggerdbl\ddagger }
     {4} { \TextOrMath \textsection\mathsection }
     {5} { \TextOrMath \textparagraph\mathparagraph }
     {6} { \TextOrMath \textbardbl\| }
     {7} { \TextOrMath {\textasteriskcentered\textasteriskcentered}{**} }
     {8} { \TextOrMath {\textdagger\textdagger}{\dagger\dagger} }
     {9} { \TextOrMath {\textdaggerdbl\textdaggerdbl}{\ddagger\ddagger} }
    }
    { \@ctrerr }
  }
  }
```

23.12 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
 \str_if_eq_x:nnTF \f@shape \itdefault \eminnershape
 {
   \str_if_eq_x:nnTF \f@shape \sldefault \eminnershape \emshape
 }
}
```

```
\DeclareTextFontCommand{\emph}{\em}
\cs_set_eq:NN \emshape \itshape
\cs_set_eq:NN \eminnershape \upshape
```

23.13 \-

\- 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{`\-}
```

23.14 Verbatims

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

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

```
\cs_new:Nn \fontspec_visible_space:
{
 \font_glyph_if_exist:NnTF \font {"2423}
  { \char"2423\scan_stop: }
  { \fontspec_visible_space_fallback: }
}
```

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

```
\cs_new:Nn \fontspec_visible_space_fallback:
{
 {
  \usefont{\g_fontspec_encoding_tl}{lmtt}{\f@series}{\f@shape}
  \textvisiblespace
 }
}
```

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

```
\group_begin:
\char_set_catcode_active:n{"20}%
\cs_gset:Npn\fontspec_print_visible_spaces:{%
\char_set_catcode_active:n{"20}%
\cs_set_eq:NN^^20\fontspec_visible_space:%
}%
\group_end:
```

\verb
\verb*

Redefine \verb to use \fontspec_print_visible_spaces:.

```
\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{\fontspec_print_visible_spaces:\@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}
  {
   \cs_set:cpn {verbatim*}
    {
    \group_begin: \@verbatim \fontspec_print_visible_spaces: \verbatim@start
    }
  }
```

This is for vanilla LaTeX.

```
  {
   \cs_set:cpn {verbatim*}
    {
     \@verbatim \fontspec_print_visible_spaces: \@sxverbatim
    }
  }
}
```

listingcont*

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

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

```
    }
  }{}
}
```

listings and fancvrb make things nice and easy:

```
\cs_set:Npn \fontspec_patch_fancyvrb:
{
 \@ifpackageloaded{fancyvrb}
  {
   \cs_set_eq:NN \FancyVerbSpace \fontspec_visible_space:
  }{}
}
\cs_set:Npn \fontspec_patch_listings:
{
 \@ifpackageloaded{listings}
  {
   \cs_set_eq:NN \lst@visiblespace \fontspec_visible_space:
  }{}
}
```

23.15 \oldstylenums

\oldstylenums
\liningnums

This command obviously needs a redefinition. And we may as well provide the reverse command.

```
\RenewDocumentCommand \oldstylenums {m}
{
 { \addfontfeature{Numbers=OldStyle} #1 }
}
\NewDocumentCommand \liningnums {m}
{
 { \addfontfeature{Numbers=Lining} #1 }
}
```

⟨/patches⟩

Part X
fontspec.cfg

```
⟨*cfg⟩

\defaultfontfeatures
 [\rmfamily,\sffamily]
 {Ligatures=TeX}

\defaultfontfeatures
 [\ttfamily]
 {WordSpace={1,0,0},
  PunctuationSpace=WordSpace}

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

% Entries here in time may be deleted.
% Please advise of any problems this causes.

\aliasfontfeatureoption{Ligatures}{Historic}{Historical}
\let\newfontinstance\newfontfamily

⟨/cfg⟩
```