The fontspec package
Font selection for XƎLATEX and LuaLATEX

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

2017/01/02 v2.5c

Contents

Part I
Getting started

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 documents `fontspec-example.tex` for a complete minimal example to get started quickly.

2.1 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 was 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.

3.1 Font encodings

The 2016 release of fontspec initiated some changes for font encodings and the loading of xunicode. The 2017 release rolls out those changes as default.

The now-default `tuenc` package option switches the NFSS font encoding to `TU`. `TU` is a new Unicode font encoding, intended for both XeTeX and LuaTeX engines, and automatically contains support for symbols covered by LaTeX's traditional `T1` and `TS1` font encodings (for example, `\%`, `\textbullet`, `\"u`, and so on). As a result, with this package option, Ross Moore's xunicode package is **not** loaded. Some new, experimental, features are now provided to customise some encoding details; see Part V on page 53 for further details.

Pre-2017 behaviour can be achieved with the `euenc` package option. This selects the `EU1` or `EU2` encoding (XeTeX/LuaTeX, resp.) and loads the xunicode package. Package authors and users who have referred explicitly to the encoding names `EU1` or `EU2` should update their code or documents. (See internal variable names described in Section 27 on page 67 for how to do this properly.)

3.2 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 shouldn't be, apply the `no-math` package option to manually suppress its behaviour here.

3.3 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.

3.4 Warnings

This package can give some 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.

4 Interaction with LaTeX 2ε and other packages

This section documents some areas of adjustment that fontspec makes to improve default behaviour with LaTeX 2ε and third-party packages.

4.1 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.

4.2 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 provides features to change the hyphenation character on a per font basis, the definition of `\-` is changed to adapt accordingly.

4.3 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*⟩`}`.

4.4 Italic small caps

`\itshape`
`\slshape`
`\scshape`

Note that this package redefines the `\itshape`, `\slshape`, and `\scshape` commands in order to allow them to select italic small caps in conjunction. With these changes, writing `\itshape\scshape` will lead to italic small caps, and `\upshape` subsequently then moves back to small caps only. `\upshape` again returns from small caps to upright regular. (And similarly for for `\slshape`. In addition, once italic small caps are selected then `\slshape` will switch to slanted small caps, and vice versa.)

4.5 Emphasis and nested emphasis

`\eminnershape` LaTeX 2$_\varepsilon$ allows you to specify the behaviour of `\emph` nested within `\emph` by setting the `\eminnershape` command. For example,

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

will produce small caps within `\emph{\emph{...}}`.

`\emfontdeclare` The fontspec package takes this idea one step further to allow arbitrary font changes (e.g., boldness) and arbitrary levels of nesting within emphasis. This is performed using the `\emfontdeclare` command, which takes a comma-separated list of font switches corresponding to increasing levels of emphasis. Two examples:

1. `\emfontdeclare{\itshape,\upshape\scshape,\itshape}` will lead to 'italics', 'small caps', then 'italic small caps' as the level of emphasis increases, as long as italic small caps are defined for the font. Note that `\upshape` is required because the font changes are cascading.

2. `\emfontdeclare{\bfseries,\fontseries{h}\selectfont,\fontseries{x}\selectfont}` could lead to (if fonts are set up correctly) 'bold', 'heavy', and 'extra bold'.

The implementation of these feature tries to be 'smart' and guess what level of emphasis to use in the case of manual font changing. This is reliable only if you use series- and/or shape- changing commands in `\emfontdeclare`. For example:

```
\emfontdeclare{\itshape,\upshape\scshape,\itshape}
...
\scshape small caps \emph{hello}
```

Here, the emphasised text 'hello' will be printed in italic small caps since `\emph` can detect that the current font shape is already in the second 'mode' of emphasis.

`\emreset` Finally, if you have so much nested emphasis that `\emfontdeclare` runs out of options, it will insert `\emreset` (by default just `\upshape`) and start again from the beginning.

Part II
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 14 on page 26, including methods for automatic scaling.

Note that while these commands all look and behave largely identically, the default setup for font loading automatically adds the `Ligatures=TeX` feature for the `\setmainfont` and `\setsansfont` commands. These defaults (and further customisations possible) are discussed in Section 8 on page 20.

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

- For general font features, see Section 14 on page 26
- For OpenType fonts, see Part IV on page 32
- For XƎTEX-only general font features, see Part VII on page 59
- For LuaTEX-only general font features, see Part VI on page 57
- For features for AAT fonts in XƎTEX, see Section 22 on page 60

5 Font selection

In both LuaTEX and XƎTEX, fonts can be selected either by 'font name' or by 'file name', but there are some differences in how each engine finds and selects fonts — don't be too surprised if a font invocation in one engine needs correction to work in the other.

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

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

```
\setmainfont{texgyrebonum-regular.otf}
\setsansfont{lmsans10-regular.otf}[Scale=MatchLowercase]
\setmonofont{Inconsolata.otf}[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
```

5.1 By font name

Fonts known to LuaTEX or XƎTEX 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. In LuaTEX, fonts found in the TEXMF tree can also be loaded by name.

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.

The 'font name' can be found in various ways, such as by looking in the name listed in a application like Font Book on Mac OS X. Alternatively, TEXLive contains the `otfinfo` command line program, which can query this information; for example:

```
  otfinfo -a `kpsewhich lmroman10-regular.otf`
```

results in '`LM Roman 10`'.

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.

5.2 By file name

XƎTEX 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 XƎTEX 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 XƎTEX 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 XƎTEX 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 using the `Path` feature without an argument:

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

My preference is to always be explicit and include the extension; this also allows fontspec to automatically identify that the font should be loaded by filename.

In previous versions of the package, the `Path` feature was also provided under the alias `ExternalLocation`, but this latter name is now deprecated and should not be used for new documents.

5.3 Querying whether a font 'exists'

`\IfFontExistsTF`{⟨*font name*⟩}{⟨*true branch*⟩}{⟨*false branch*⟩}

The conditional `\IfFontExistsTF` is provided to test whether the ⟨*font name*⟩ exists or is loadable. If it is, the ⟨*true branch*⟩ code is executed; otherwise, the ⟨*false branch*⟩ code is.

This command can be slow since the engine may resort to scanning the filesystem for a missing font. Nonetheless, it has been a popular request for users who wish to define 'fallback fonts' for their documents for greater portability.

In this command, the syntax for the ⟨*font name*⟩ is a restricted/simplified version of the font loading syntax used for `\fontspec` and so on. Fonts to be loaded by filename are detected by the presence of an appropriate extension (`.otf`, etc.), and paths should be included inline. E.g.:

```
\IfFontExistsTF{cmr10}{T}{F}
\IfFontExistsTF{Times New Roman}{T}{F}
\IfFontExistsTF{texgyrepagella-regular.otf}{T}{F}
\IfFontExistsTF{/Users/will/Library/Fonts/CODE2000.TTF}{T}{F}
```

The `\IfFontExistsTF` command is a synonym for the programming interface function `\fontspec_font_if_exist:nTF` (Section 27 on page 67).

6 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 22.4 on page 61.

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,

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
```

```
\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}
```

6.1 More control over font shape selection

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

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

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} \\
```

6.1.1 Small caps and slanted font shapes

When a font family has both 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.

Pre-OpenType, it was common for font families to be distributed with 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:

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

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

```
\setmainfont{ <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.

All of the bold, italic, and small caps fonts can be loaded with different font features from the main font. See Section 11 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,

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

6.2 Specifically choosing the NFSS family

In LaTeX's NFSS, font families are defined with names such as 'ppl' (Palatino), 'lmr' (Latin 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)'. Please note that should not rely on the name that is generated.

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 27 on page 67).

6.3 Choosing additional NFSS font faces

LaTeX's font selection scheme (NFSS) 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 additional ⟨*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}{\updefault}{ 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 an upright shape (`\updefault`, as shown above) will attempt to find a matching small caps feature and define that face as well. Conversely, a font face defined in terms of a non-standard font shape will not.

There are some standards for choosing shape and series codes; the LaTeX 2ε 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.

6.4 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

[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}`

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⟩]
```

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}
\setmainfont{Optima}
\setmathrm{Optima}
\setboldmathrm[BoldFont={Optima ExtraBlack}]{Optima Bold}
```

These commands are compatible with the unicode-math package. Having said that, unicode-math also defines a more general way of defining fonts to use in maths mode, so you can ignore this subsection if you're already using that package.

7 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, because the most important detail — the name of the font — was tucked away at the end. 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, so either input method works.
2. In fact, you can write

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

   if you really felt like it and both sets of features would be applied.
3. 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}%        \fontspec{myfont.otf}[
     [Color=red]                    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.

Part III

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 part discusses these options.

8 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.

```
\defaultfontfeatures[texgyreadventor-regular.otf]{Color=blue}
\setmainfont{texgyreadventor-regular.otf}% will be blue
```

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{texgyreadventor-regular.otf}% will use standard TeX ligatures
```

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

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

```
\fontspec{texgyreadventor-regular.otf}
Some default text 0123456789 \\
\defaultfontfeatures{
   Numbers=OldStyle, Color=888888
}
\fontspec{texgyreadventor-regular.otf}
Now grey, with old-style figures:
0123456789
```

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{texgyreadventor-regular.otf}% 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.

9 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`'.[5]

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.

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

10 Working with the currently selected features

`\IfFontFeatureActiveTF{⟨font feature⟩}{⟨true code⟩}{⟨false code⟩}`

This command queries the currently selected font face and executes the appropriate branch based on whether the ⟨*font feature*⟩ as specified by fontspec is currently active.

For example, the following will print 'True':

```
\setmainfont{texgyrepagella-regular.otf}[Numbers=OldStyle]
\IfFontFeatureActiveTF{Numbers=OldStyle}{True}{False}
```

Note that there is no way for fontspec to know what the default features of a font will be. For example, by default the `texgyrepagella` fonts use lining numbers. But in the following example, querying for lining numbers returns false since they have not been explicitly requested:

```
\setmainfont{texgyrepagella-regular.otf}
\IfFontFeatureActiveTF{Numbers=Lining}{True}{False}
```

Please note: At time of writing this function only supports OpenType fonts; AAT/Graphite fonts under the XeTeX engine are not supported.

`\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. If you attempt to change an already-selected feature, fontspec will try to deactivate any features that clash with the new ones. *E.g.*, the following two invocations are mutually exclusive:

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

Since `Numbers=Lining` comes last, it takes precedence and deactivates the call `Numbers=OldStyle`.

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

10.1 Priority of feature selection

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

Example 6: A demonstration of the \addfontfeatures command.

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

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

```
\fontspec{texgyreadventor-regular.otf}%
         [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}}
```

11 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 7: Features for, say, just italics.

Don't Ask Victoria!
Don't Ask Victoria!

```
\fontspec{EBGaramond12-Regular.otf}%
  [ItalicFont=EBGaramond12-Italic.otf]
\itshape Don't Ask Victoria! \\
\addfontfeature{ItalicFeatures={Style=Swash}}
Don't Ask Victoria! \\
```

Example 8: An example of setting the `SmallCapsFeatures` separately for each font shape.

```
\fontspec{texgyretermes}[
    Extension = {.otf},
    UprightFont = {*-regular}, ItalicFont = {*-italic},
    BoldFont = {*-bold}, BoldItalicFont = {*-bolditalic},
    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}
```

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

12 Selecting fonts from TrueType Collections (TTC files)

TrueType Collections are multiple fonts contained within a single file. Each font within a collection must be explicitly chosen using the `FontIndex` command. Since TrueType Collections are often used to contain the italic/bold shapes in a family, fontspec automatically selects the italic, bold, and bold italic fontfaces from the same file. For example, to load the macOS system font Optima:

```
\setmainfont{Optima.ttc}[
  Path = /System/Library/Fonts/ ,
  UprightFeatures    = {FontIndex=0} ,
  BoldFeatures       = {FontIndex=1} ,
  ItalicFeatures     = {FontIndex=2} ,
  BoldItalicFeatures = {FontIndex=3} ,
]
```

Support for TrueType Collections has only been tested in XƎTEX, but should also work with an up-to-date version of LuaTEX and the luaotfload package.

13 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 14.6 on page 30.

To be precise, the `Size` sub-feature accepts arguments in the form shown in Table 1 on the next page. 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!

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

```
UprightFeatures =
 {
```

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

Small

Normal size

Large

```
\fontspec{texgyrechorus-mediumitalic.otf}[
  SizeFeatures={
    {Size={-8}, Font=texgyrebonum-italic.otf, Color=AA0000},
    {Size={8-14}, Color=00AA00},
    {Size={14-}, Color=0000AA}} ]

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

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

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

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

Suggestions on simplifying this interface welcome.

14 Font independent options

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

14.1 Colour

Color (or Colour) uses font specifications to set the colour of the text. You should think of this as the literal glyphs of the font being coloured in a certain way. Notably, this mechanism is different to that of the color/xcolor/hyperref/etc. packages, and in fact using fontspec commands to set colour will prevent your text from changing colour using those packages at all! For example, if you set the colour in a \setmainfont command, \color{...} and related commands, including hyperlink colouring, will no longer have any effect on text in this font.) Therefore, fontspec's colour commands are best used to set explicit colours in specific situations, and the xcolor package is recommended for more general colour functionality.

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} ...
```

Example 10: Selecting colour with transparency.

WSPR

```
\fontsize{48}{48}
\fontspec{texgyrebonum-bold.otf}
{\addfontfeature{Color=FF000099}W}\kern-0.4ex
{\addfontfeature{Color=0000FF99}S}\kern-0.4ex
{\addfontfeature{Color=DDBB2299}P}\kern-0.5ex
{\addfontfeature{Color=00BB3399}R}
```

```
\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:

```
\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.

14.2 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.

Example 11: Automatically calculated scale values.

The perfect match is hard to find.
LOGOFONT

```
\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
```

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 3.3 on page 7) for this behaviour to be effected in your own documents automatically. (Also see Section 8 on page 20 for more information on setting font defaults.)

14.3 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}.)

Note that TEX's optimisations in how it loads fonts means that you cannot use this feature in \addfontfeatures.

14.4 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.

Note that TEX's optimisations in how it loads fonts means that you cannot use this feature in \addfontfeatures.

14.5 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.

Note that TEX's optimisations in how it loads fonts means that you cannot use this feature in \addfontfeatures.

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{texgyretermes-regular.otf}
Some text for our example to take
up some space, and to demonstrate
the default interword space.
\bigskip

\fontspec{texgyretermes-regular.otf}%
  [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{texgyreschola-regular.otf}
 Letters, Words. Sentences.          \par
\fontspec{texgyreschola-regular.otf}[PunctuationSpace=2]
 Letters, Words. Sentences.          \par
\fontspec{texgyreschola-regular.otf}[PunctuationSpace=0]
 Letters, Words. Sentences.
```

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

14.6 Optical font sizes

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

OpenType fonts with optical scaling will exist in several discrete sizes, and these will be selected by XeTeX 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 13 on page 25) 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}}}
         ]
```

14.7 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 17. 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.

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}
```

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
```

Example 17: 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
```

14.8 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 18.

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 is not generally used for lowercase text in modern typesetting but does have historic precedent in a variety of situations. In particular, small amounts of letter spacing can be very useful, when setting small caps or all caps titles. Also see the OpenType Uppercase option of the Letters feature (Section 16.2 on page 34).

Example 18: The LetterSpace feature.

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

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

Part IV
OpenType

15 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 font features. When the user applies a feature — for example, small capitals — to a run of text, the code inside the font makes appropriate substitutions 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 Indic scripts, 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 localises the shapes of letters.

There are a large group of OpenType features, designed to support high quality typography a multitude of languages and writing 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 16.

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.

15.1 How to select font features

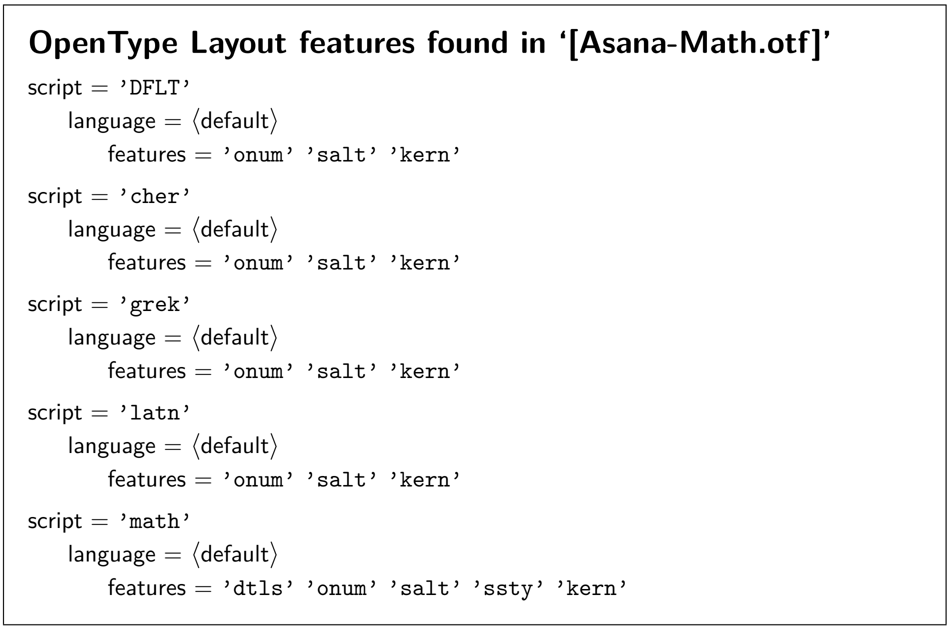
Font features are selected by a series of ⟨*feature*⟩=⟨*option*⟩ selections. Features are (usually) grouped logically; for example, all font features relating to ligatures are accessed by writing `Ligatures={...}` with the appropriate argument(s), which could be `TeX`, `Rare`, etc., as shown below in 16.1.1.

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 XƎTEX 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 3.4 on page 8 these warnings can be suppressed by selecting the [quiet] package option.

15.2 How do I know what font features are supported by my fonts?

Although I've long desired to have a feature within fontspec to display the OpenType features within a font, it's never been high on my priority list. One reason for that is the existence of the document `opentype-info.tex`, which is available on CTAN or typing `kpsewhich opentype-info.tex` in a Terminal window. Make a copy of this file and place it somewhere convenient. Then open it in your regular TEX editor and change the font name to the font you'd like to query; after running through plain XƎTEX, the output PDF will look something like this:



OpenType Layout features found in '[Asana-Math.otf]'

script = 'DFLT'
    language = ⟨default⟩
        features = 'onum' 'salt' 'kern'

script = 'cher'
    language = ⟨default⟩
        features = 'onum' 'salt' 'kern'

script = 'grek'
    language = ⟨default⟩
        features = 'onum' 'salt' 'kern'

script = 'latn'
    language = ⟨default⟩
        features = 'onum' 'salt' 'kern'

script = 'math'
    language = ⟨default⟩
        features = 'dtls' 'onum' 'salt' 'ssty' 'kern'

I intentionally picked a font that by design needs few font features; 'regular' text fonts such as Latin Modern Roman contain many more, and I didn't want to clutter up the document too much. You'll then need to cross-check the OpenType feature tags with the 'logical' names used by fontspec.

otfinfo Alternatively, and more simply, you can use the command line tool `otfinfo`, which is distributed with TEXLive. Simply type in a Terminal window, say:

```
otfinfo -f `kpsewhich lmromandunh10-oblique.otf`
```

which results in:

```
aalt        Access All Alternates
cpsp        Capital Spacing
dlig        Discretionary Ligatures
frac        Fractions
```

kern	Kerning
liga	Standard Ligatures
lnum	Lining Figures
onum	Oldstyle Figures
pnum	Proportional Figures
size	Optical Size
tnum	Tabular Figures
zero	Slashed Zero

16 OpenType font features

There are a finite set of OpenType font features, and fontspec provides an interface to around half of them. Full documentation will be presented in the following sections, including how to enable and disable individual features, and how they interact.

A brief reference is provided (Table 2 on the next page) but note that this is an incomplete listing — only the 'enable' keys are shown, and where alternative interfaces are provided for convenience only the first is shown. (E.g., `Numbers=OldStyle` is the same as `Numbers=Lowercase`.)

For completeness, the complete list of OpenType features not provided with a fontspec interface is shown in Table 3 on page 36. Features omitted are partially by design and partially by oversight; for example, the `aalt` feature is largely useless in TeX since it is designed for providing a textscgui interface for selecting 'all alternates' of a glyph. Others, such as optical bounds for example, simply haven't yet been considered due to a lack of fonts available for testing. Suggestions welcome for how/where to add these missing features to the package.

16.1 Tag-based features

16.1.1 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 4. A demonstration with the Linux Libertine fonts[6] is shown in Example 19.

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 21.1 on page 59) but for consistency `Ligatures=TeX` will perform the same function as `Mapping=tex-text`.

16.2 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`.

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

Table 2: Summary of OpenType features in fontspec, alphabetic by feature tag.

abvm	Diacritics = AboveBase	*Above-base Mark Positioning*
afrc	Fractions = Alternate	*Alternative Fractions*
blwm	Diacritics = BelowBase	*Below-base Mark Positioning*
calt	Contextuals = Alternate	*Contextual Alternates*
case	Letters = Uppercase	*Case-Sensitive Forms*
clig	Ligatures = Contextual	*Contextual Ligatures*
cpsp	Kerning = Uppercase	*Capital Spacing*
cswh	Contextuals = Swash	*Contextual Swash*
cvNN	CharacterVariant = N : M	*Character Variant N*
c2pc	Letters = UppercasePetiteCaps	*Petite Capitals From Capitals*
c2sc	Letters = UppercaseSmallCaps	*Small Capitals From Capitals*
dlig	Ligatures = Rare	*Discretionary Ligatures*
dnom	VerticalPosition = Denominator	*Denominators*
expt	CJKShape = Expert	*Expert Forms*
falt	Contextuals = LineFinal	*Final Glyph on Line Alternates*
fina	Contextuals = WordFinal	*Terminal Forms*
frac	Fractions = On	*Fractions*
fwid	CharacterWidth = Full	*Full Widths*
halt	CharacterWidth = AlternateHalf	*Alternate Half Widths*
hist	Style = Historic	*Historical Forms*
hkna	Style = HorizontalKana	*Horizontal Kana Alternates*
hlig	Ligatures = Historic	*Historical Ligatures*
hwid	CharacterWidth = Half	*Half Widths*
init	Contextuals = WordInitial	*Initial Forms*
ital	Style = Italic	*Italics*
jp78	CJKShape = JIS1978	*JIS78 Forms*
jp83	CJKShape = JIS1983	*JIS83 Forms*
jp90	CJKShape = JIS1990	*JIS90 Forms*
jp04	CJKShape = JIS2004	*JIS2004 Forms*
kern	Kerning = On	*Kerning*
liga	Ligatures = Common	*Standard Ligatures*
lnum	Numbers = Uppercase	*Lining Figures*
mark	Diacritics = MarkToBase	*Mark Positioning*
medi	Contextuals = Inner	*Medial Forms*
mkmk	Diacritics = MarkToMark	*Mark to Mark Positioning*
nalt	Annotation = N	*Alternate Annotation Forms*
nlck	CJKShape = NLC	*NLC Kanji Forms*
numr	VerticalPosition = Numerator	*Numerators*
onum	Numbers = Lowercase	*Oldstyle Figures*
ordn	VerticalPosition = Ordinal	*Ordinals*
ornm	Ornament = N	*Ornaments*
palt	CharacterWidth = AlternateProportional	*Proportional Alternate Widths*
pcap	Letters = PetiteCaps	*Petite Capitals*
pkna	Style = ProportionalKana	*Proportional Kana*
pnum	Numbers = Proportional	*Proportional Figures*
pwid	CharacterWidth = Proportional	*Proportional Widths*
qwid	CharacterWidth = Quarter	*Quarter Widths*
rand	Letters = Random	*Randomize*
rlig	Ligatures = Required	*Required Ligatures*
ruby	Style = Ruby	*Ruby Notation Forms*
salt	Alternate = N	*Stylistic Alternates*
sinf	VerticalPosition = ScientificInferior	*Scientific Inferiors*
smcp	Letters = SmallCaps	*Small Capitals*
smpl	CJKShape = Simplified	*Simplified Forms*
ssNN	StylisticSet = N	*Stylistic Set N*
ssty	Style = MathScript	*Math script style alternates*
subs	VerticalPosition = Inferior	*Subscript*
sups	VerticalPosition = Superior	*Superscript*
swsh	Style = Swash	*Swash*
titl	Style = TitlingCaps	*Titling*
tnum	Numbers = Monospaced	*Tabular Figures*
trad	CJKShape = Traditional	*Traditional Forms*
twid	CharacterWidth = Third	*Third Widths*
unic	Letters = Unicase	*Unicase*
valt	Vertical = AlternateMetrics	*Alternate Vertical Metrics*
vert	Vertical = Alternates	*Vertical Writing*
vhal	Vertical = HalfMetrics	*Alternate Vertical Half Metrics*
vkna	Style = VerticalKana	*Vertical Kana Alternates*
vkrn	Vertical = Kerning	*Vertical Kerning*
vpal	Vertical = ProportionalMetrics	*Proportional Alternate Vertical Metrics*
vrt2	Vertical = RotatedGlyphs	*Vertical Alternates and Rotation*
vrtr	Vertical = AlternatesForRotation	*Vertical Alternates for Rotation*
zero	Numbers = SlashedZero	*Slashed Zero*

Table 3: List of unsupported OpenType features.

AALT	*Access All Alternates*
ABVF	*Above-base Forms*
ABVS	*Above-base Substitutions*
AKHN	*Akhands*
BLWF	*Below-base Forms*
BLWS	*Below-base Substitutions*
CCMP	*Glyph Composition / Decomposition*
CFAR	*Conjunct Form After Ro*
CJCT	*Conjunct Forms*
CPCT	*Centered CJK Punctuation*
CURS	*Cursive Positioning*
DIST	*Distances*
DTLS	*Dotless Forms*
FIN2	*Terminal Forms #2*
FIN3	*Terminal Forms #3*
FLAC	*Flattened accent forms*
HALF	*Half Forms*
HALN	*Halant Forms*
HNGL	*Hangul*
HOJO	*Hojo Kanji Forms*
ISOL	*Isolated Forms*
JALT	*Justification Alternates*
LFBD	*Left Bounds*
LJMO	*Leading Jamo Forms*
LOCL	*Localized Forms*
LTRA	*Left-to-right alternates*
LTRM	*Left-to-right mirrored forms*
MED2	*Medial Forms #2*
MGRK	*Mathematical Greek*
MSET	*Mark Positioning via Substitution*
NUKT	*Nukta Forms*
OPBD	*Optical Bounds*
PREF	*Pre-Base Forms*
PRES	*Pre-base Substitutions*
PSTF	*Post-base Forms*
PSTS	*Post-base Substitutions*
RCLT	*Required Contextual Alternates*
RKRF	*Rakar Forms*
RPHF	*Reph Forms*
RTBD	*Right Bounds*
RTLA	*Right-to-left alternates*
RTLM	*Right-to-left mirrored forms*
RVRN	*Required Variation Alternates*
SIZE	*Optical size*
STCH	*Stretching Glyph Decomposition*
TJMO	*Trailing Jamo Forms*
TNAM	*Traditional Name Forms*
VATU	*Vattu Variants*
VJMO	*Vowel Jamo Forms*

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

Feature		Option	Tag
Ligatures	=	Required	`rlig` †
		Common	`liga` †
		Contextual	`clig` †
		Rare/Discretionary	`dlig` †
		Historic	`hlig` †
		TeX	`tlig` †
		ResetAll	

† These feature options can be disabled with `..Off` variants, and reset to default state (neither explicitly on nor off) with `..Reset`.

Example 19: An example of the `Ligatures` feature.

strict → ﬆrict
wurtzite → wurꜩite
ﬁreﬂy → firefly

```
\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}
```

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

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

† These feature options can be disabled with `..Off` variants, and reset to default state (neither explicitly on nor off) with `..Reset`.

Petite caps are smaller than small caps. SmallCaps and PetiteCaps turn lowercase letters into the smaller caps letters, whereas the Uppercase... options turn the capital letters into the smaller caps (good, *e.g.*, for applying to already uppercase acronyms like 'NASA'). This difference is shown in Example 20. 'Unicase' is a weird hybrid of upper and lower case letters.

Note that the Uppercase option will (probably) not actually map letters to uppercase.[7] It is designed to select various uppercase forms for glyphs such as accents and dashes, such as shown in Example 21; 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 16.5 on page 43).

16.2.1 Numbers

The Numbers feature defines how numbers will look in the selected font, accepting options shown in Table 6.

The synonyms Uppercase and Lowercase are equivalent to Lining and OldStyle, respectively. The differences have been shown previously in Section 10 on page 22. 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 22.

The Arabic option (with tag anum) maps regular numerals to their Arabic script or Persian equivalents based on the current Language setting (see Section 16.9 on page 48). This option is based on a LuaTEX feature of the luaotfload package, not an OpenType feature. (Thus, this feature is unavailable in XETEX.)

16.2.2 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 7.

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.

Example 20: Small caps from lowercase or uppercase letters.

THIS SENTENCE NO VERB
THIS SENTENCE no verb

```
\fontspec{texgyreadventor-regular.otf}[Letters=SmallCaps]
 THIS SENTENCE no verb                \\
\fontspec{texgyreadventor-regular.otf}[Letters=UppercaseSmallCaps]
 THIS SENTENCE no verb
```

Example 21: 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 6: Options for the OpenType font feature 'Numbers'.

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

† These feature options can be disabled with `..Off` variants, and reset to default state (neither explicitly on nor off) with `..Reset`.

Example 22: The effect of the SlashedZero option.

0123456789 0123456789

```
\fontspec[Numbers=Lining]{texgyrebonum-regular.otf}
 0123456789
\fontspec[Numbers=SlashedZero]{texgyrebonum-regular.otf}
 0123456789
```

Table 7: Options for the OpenType font feature 'Contextuals'.

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

† These feature options can be disabled with `..Off` variants, and reset to default state (neither explicitly on nor off) with `..Reset`.

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

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

† These feature options can be disabled with `..Off` variants, and reset to default state (neither explicitly on nor off) with `..Reset`.

16.2.3 Vertical Position

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

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

The realscripts package (which is also loaded by xltxtra for XƎTEX) 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.

16.2.4 Fractions

For OpenType fonts use a regular text slash to create fractions, but the `Fraction` feature must be explicitly activated. Some (Asian fonts predominantly) also provide for the `Alternate` feature. These are both shown in Example 24.

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

Example 23: The `VerticalPosition` feature.

Superior: ^{1234567890}
Numerator: ^{12345}
Denominator: $_{12345}$
Scientific Inferior: $_{12345}$

```
\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
```

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

Feature		Option	Tag	
Fractions	=	On	+frac	
		Off	-frac	
		Reset		
		Alternate	afrc	†
		ResetAll		

† These feature options can be disabled with ..Off variants, and reset to default state (neither explicitly on nor off) with ..Reset.

Example 24: The Fractions feature.

1/2 1/4 5/6 13579/24680
½ ¼ ⅚ 13579/24680
$\frac{1}{2}$ $\frac{1}{4}$ $\frac{5}{6}$ 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 \\
```

16.3 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 25 and Example 26 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 27 and Example 28; 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 29: the horizontal style is slightly wider.

Example 25: 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
```

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

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

† These feature options can be disabled with `..Off` variants, and reset to default state (neither explicitly on nor off) with `..Reset`.

Example 26: 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 27: 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 28: 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 29: 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 }
```

16.4 Diacritics

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

16.5 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 16.2 on page 34, the `Uppercase` option will add a small amount of tracking between uppercase letters, seen in Example 30, which uses the Romande fonts[8] (thanks to Clea F. Rees for the suggestion). The `Uppercase` option acts separately to the regular kerning controlled by the `On`/`Off` options.

16.6 Character width

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

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

The same situation occurs with numbers, which are provided in increasingly illegible compressed forms seen in Example 32.

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

Feature		Option	Tag	
Diacritics	=	MarkToBase	`mark`	†
		MarkToMark	`mkmk`	†
		AboveBase	`abvm`	†
		BelowBase	`blwm`	†
		ResetAll		

† These feature options can be disabled with `..Off` variants, and reset to default state (neither explicitly on nor off) with `..Reset`.

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

Feature		Option	Tag	
Kerning	=	On	+kern	
		Off	-kern	
		Reset		
		Uppercase	cpsp	†
		ResetAll		

† These feature options can be disabled with `..Off` variants, and reset to default state (neither explicitly on nor off) with `..Reset`.

Example 30: 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
```

Table 13: 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	†
		ResetAll		

† These feature options can be disabled with `..Off` variants, and reset to default state (neither explicitly on nor off) with `..Reset`.

Example 31: 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 32: Numbers can be compressed significantly.

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

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

Table 14: 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`

† These feature options can be disabled with `..Off` variants, and reset to default state (neither explicitly on nor off) with `..Reset`.

16.6.1 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.

16.7 Vertical typesetting

OpenType provides a plethora of features for accommodating the varieties of possibilities needed for vertical typesetting (CJK and others). No capabilities for achieving such vertical typesetting are provided by fontspec, however; please get in touch if there are improvements that could be made.

16.8 Numeric features

16.8.1 Stylistic Set variations — `ssNN`

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 34 and Example 35; 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 23 on page 65 for a way to assign names to stylistic sets, which should be done on a per-font basis.

16.8.2 Character Variants — `cvNN`

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`.

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

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

Example 33: Different standards for CJK ideograph presentation.

唖噛躯 妍并訝
唖噛躯 妍并訝
啞嚙軀 妍并訝

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

Table 15: Options for the OpenType font feature 'Vertical'.

Feature		Option	Tag	
Vertical	=	RotatedGlyphs	`vrt2`	†
		AlternatesForRotation	`vrtr`	†
		Alternates	`vert`	†
		KanaAlternates	`vkna`	†
		Kerning	`vkrn`	†
		AlternateMetrics	`valt`	†
		HalfMetrics	`vhal`	†
		ProportionalMetrics	`vpal`	†
		ResetAll		

† These feature options can be disabled with `..Off` variants, and reset to default state (neither explicitly on nor off) with `..Reset`.

Example 34: Insular letterforms, as used in medieval Northern Europe, for the Junicode font accessed with the `StylisticSet` feature.

Insular forms.
Inꞅulaꞃ ꝼoꞃmꞅ.

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

Example 35: 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. \\
```

For each character that can be varied, it is possible to select among possible options for that particular glyph. For example, in Example 36 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 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 37. (Unlike stylistic alternates, say.)

Note that the indexing starts from zero.

16.8.3 Alternates — salt

The Alternate feature, alias StylisticAlternates, is used to access alternate font glyphs when variations exist in the font, such as in Example 38. It uses a numerical selection, starting from zero, that will be different for each font. Note that the Style=Alternate option is equivalent to Alternate=0 to access the default case.

Note that the indexing starts from zero. With the LuaTEX engine, Alternate=Random selects a random alternate.

See Section 23 on page 65 for a way to assign names to alternates if desired.

16.8.4 Annotation — nalt

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 39. Note that the indexing starts from zero.

16.8.5 Ornament — ornm

Ornaments are selected with the Ornament feature (OpenType feature ornm), selected numerically such as for the Annotation feature. If you know of an Open Source font that supports this feature, let me know and I'll add an example.

16.9 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 16 on page 51 and Table 17 on page 52, respectively. When a script or language is requested that is not supported by the current font, a warning is printed in the console output.

Because these font features can change which features are able to be selected for the font, they are automatically selected by fontspec before all others and, if XƎTEX is

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

Example 36: 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
```

Example 37: 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
```

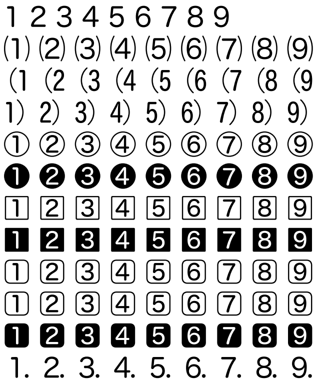
Example 38: The Alternate feature.

A & h

A & h

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

Example 39: Annotation forms for OpenType fonts.



```
\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
```

being used, will specifically select the OpenType renderer for this font, as described in Section 21.2 on page 59.

See Section 24 on page 66 for methods to create new Script or Language options if required.

16.9.1 Script and Language examples

In the examples shown in Example 40, the Code2000 font[11] 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.

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

Example 40: An example of various Scripts and Languages.

العربي	العربي
हनि्दी	हिन्दी
লখে	লেখ
મર્યાદા-સૂચક નવિેદન	મર્યાદા-સૂચક નિવેદન
നമ്മുടെ പാരബര്യ	നമ്മുടെ പാരബര്യ
ਆਦ ਿਸਚੁ ਜੁਗਾਦ ਿਸਚੁ	ਆਦਿ ਸਚੁ ਜੁਗਾਦਿ ਸਚੁ
தமிழ் தடேி	தமிழ் தேடி
רְדָתָּה	רְדָתָּה
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}
```

Table 16: Defined Scripts for OpenType fonts. Aliased names are shown in adjacent positions marked with red pilcrows (¶).

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

Table 17: Defined Languages for OpenType fonts. Aliased names are shown in adjacent positions marked with red pilcrows (¶).

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

Part V

Commands for accents and symbols ('encodings')

The functionality described in this section is experimental.

In the pre-Unicode era, significant work was required by LaTeX to ensure that input characters in the source could be interpreted correctly depending on file encoding, and that glyphs in the output were selected correctly depending on the font encoding. With Unicode, we have the luxury of a single file and font encoding that is used for both input and output.

While this may provide some illusion that we could get away simply with typing Unicode text and receive correct output, this is not always the case. For a start, hyphenation in particular is language-specific, so tags should be used when switch between languages in a document. The babel and polyglossia packages both provide features for this.

Multilingual documents will often use different fonts for different languages, not just for style, but for the more pragmatic reason that fonts do not all contain the same glyphs. (In fact, only test fonts such as Code2000 provide anywhere near the full Unicode coverage.) Indeed, certain fonts may be perfect for a certain application but miss a handful of necessary diacritics or accented letters. In these cases, fontspec can leverage the font encoding technology built into LaTeX2 to provide on a per-font basis either provide fallback options or error messages when a desired accent or symbol is not available. However, at present these features can only be provided for input using LaTeX commands rather than Unicode input; for example, typing `` \`e `` instead of è or `\textcopyright` instead of © in the source file.

The most widely-used encoding in LaTeX 2ε was `T1` with companion '`TS1`' symbols provided by the textcomp package. These encodings provided glyphs to typeset text in a variety of western European languages. As with most legacy LaTeX 2ε input methods, accents and symbols were input using encoding-dependent commands such as `` \`e `` as described above. As of 2017, in LaTeX 2ε on XeTeX and LuaTeX, the default encoding is `TU`, which uses Unicode for input and output. The `TU` encoding provides appropriate encoding-dependent definitions for input commands to match the coverage of the `T1`+`TS1` encodings. Wider coverage is not provided by default since (a) each font will provide different glyph coverage, and (b) it is expected that most users will be writing with direct Unicode input.

For those users who do need finer-grained control, fontspec provides an interface for a more extensible system.

17 A new Unicode-based encoding from scratch

Let's say you need to provide support for a document originally written with fonts in the `OT2` encoding, which contains encoding-dependent commands for Cyrillic letters. An example from the `OT2` encoding definition file (`ot2enc.def`) reads:

```
\DeclareTextSymbol{\CYRIE}{OT2}{5}
\DeclareTextSymbol{\CYRDJE}{OT2}{6}
\DeclareTextSymbol{\CYRTSHE}{OT2}{7}
\DeclareTextSymbol{\cyrnje}{OT2}{8}
\DeclareTextSymbol{\cyrlje}{OT2}{9}
\DeclareTextSymbol{\cyrdzhe}{OT2}{10}
```

To recreate this encoding in a form suitable for fontspec, create a new file named, say, `fontrange-cyr.def` and populate it with

```
...
\DeclareTextSymbol{\CYRIE}  {\LastDeclaredEncoding}{"0404}
\DeclareTextSymbol{\CYRDJE} {\LastDeclaredEncoding}{"0402}
\DeclareTextSymbol{\CYRTSHE}{\LastDeclaredEncoding}{"040B}
\DeclareTextSymbol{\cyrnje} {\LastDeclaredEncoding}{"045A}
\DeclareTextSymbol{\cyrlje} {\LastDeclaredEncoding}{"0459}
\DeclareTextSymbol{\cyrdzhe}{\LastDeclaredEncoding}{"045F}
...
```

The numbers `"0404`, `"0402`, …, are the Unicode slots (in hexadecimal) of each glyph respectively. The fontspec package provides a number of shorthands to simplify this style of input; in this case, you could also write

```
\EncodingSymbol{\CYRIE}{"0404}
...
```

To use this encoding in a fontspec font, you would first add this to your preamble:

```
\DeclareUnicodeEncoding{unicyr}{
  \input{fontrange-cyr.def}
}
```

Then follow it up with a font loading call such as

```
\setmainfont{...}[NFSSEncoding=unicyr]
```

The first argument `unicyr` is the name of the 'encoding' to use in the font family. (There's nothing special about the name chosen but it must be unique.) The second argument to `\DeclareUnicodeEncoding` also allows adjustments to be made for per-font changes. We'll cover this use case in the next section.

18 Adjusting a pre-existing encoding

There are three reasons to adjust a pre-existing encoding: to add, to remove, and to redefine some symbols, letters, and/or accents.

When adding symbols, etc., simply write

```
\DeclareUnicodeEncoding{unicyr}{
  \input{tuenc.def}
  \input{fontrange-cyr.def}
  \EncodingSymbol{\textruble}{"20BD}
}
```

Of course if you consistently add a number of symbols to an encoding it would be a good idea to create a new `fontrange-XX.def` file to suit your needs.

When removing symbols, use the `\UndeclareSymbol{`⟨*cmd*⟩`}` command. For example, if you a loading a font that you know is missing, say, the interrobang (not that unusual a situation), you might write:

```
\DeclareUnicodeEncoding{nobang}{
  \input{tuenc.def}
  \UndeclareSymbol\textinterrobang
}
```

Provided that you use the command `\textinterrobang` to typeset this symbol, it will appear in fonts with the default encoding, while in any font loaded with the `nobang` encoding an attempt to access the symbol will either use the default fallback definition or return an error, depending on the symbol being undeclared.

The third use case is to redefine a symbol or accent. The most common use case in this scenario is to adjust a specific accent command to either fine-tune its placement or to 'fake' it entirely. For example, the underdot diacritic is used in typeset Sanskrit, but it is not necessarily included as an accent symbol is all fonts. By default the underdot is defined in `TU` as:

```
\EncodingAccent{\d}{"0323}
```

For fonts with a missing (or poorly-spaced) `"0323` accent glyph, the 'traditional' TeX fake accent construction could be used instead:

```
\DeclareUnicodeEncoding{fakeacc}{
  \input{tuenc.def}
  \EncodingCommand{\d}[1]{%
    \hmode@bgroup
      \o@lign{\relax#1\crcr\hidewidth\ltx@sh@ft{-1ex}.\hidewidth}%
    \egroup
  }
}
```

This would be set up in a document as such:

```
\newfontfamily\sanskitfont{CharisSIL}
\newfontfamily\titlefont{Posterama}[NFSSEncoding=fakeacc]
```

Then later in the document, no additional work is needed:

```
...{\titlefont   kalita\d m}...  % <- uses fake accent
...{\sanskitfont kalita\d m}...  % <- uses real accent
```

To reiterate from above, typing this input with Unicode text ('`kalitaṃ`') will bypass this encoding mechanism and you will receive only what is contained literally within the font.

19 Summary of commands

The LaTeX 2ε kernel provides the following font encoding commands suitable for Unicode encodings:

```
\DeclareTextCommand{⟨command⟩}{⟨encoding⟩}[⟨num⟩][⟨default⟩]{⟨code⟩}
\DeclareUnicodeAccent{⟨command⟩}{⟨encoding⟩}{⟨slot⟩}
\DeclareTextSymbol{⟨command⟩}{⟨encoding⟩}{⟨slot⟩}
\DeclareTextComposite{⟨command⟩}{⟨encoding⟩}{⟨letter⟩}{⟨slot⟩}
\DeclareTextCompositeCommand{⟨command⟩}{⟨encoding⟩}{⟨letter⟩}{⟨code⟩}
\UndeclareTextCommand{⟨command⟩}{⟨encoding⟩}
```

See `fntguide.pdf` for full documentation of these. As shown above, the following shorthands a provided by fontspec to simplify the process of defining Unicode font range encodings:

```
\EncodingCommand{⟨command⟩}[⟨num⟩][⟨default⟩]{⟨code⟩}
\EncodingAccent{⟨command⟩}{⟨code⟩}
\EncodingSymbol{⟨command⟩}{⟨code⟩}
\EncodingComposite{⟨command⟩}{⟨letter⟩}{⟨slot⟩}
\EncodingCompositeCommand{⟨command⟩}{⟨letter⟩}{⟨code⟩}
\UndeclareSymbol{⟨command⟩}
\UndeclareComposite{⟨command⟩}{⟨letter⟩}
```

Despite its name, `\UndeclareSymbol` can be used for commands defined by all three of `\EncodingCommand`, `\EncodingAccent`, and `\EncodingSymbol`.

Part VI
LuaTEX-only font features

20 Custom font features

Pre-2016, it was possible to load an OpenType font feature file to define new OpenType features for a selected font. This facility was particularly useful to implement custom substitutions, for example. As of TEXLive 2016, LuaTEX/luaotfload no longer supports this feature, but provides its own internal mechanisms for an equivalent interface.

Any documents using 'feature file' options will need to transition to the new interface. Figure 1 shows an example. Please refer to the LuaTEX/luaotfload documentation for more details.

Figure 1: An example of custom font features.

```
\documentclass{article}
\usepackage{fontspec}
\directlua{
    fonts.handlers.otf.addfeature {
        name = "oneb",
        {
            type = "substitution",
            data = {
                ["1"] = "one.ss01",
            },
        },
        "feature oneb for vollkorn font",
    }
}
\setmainfont{Vollkorn-Regular.otf}[RawFeature=+oneb]
\begin{document}
1234567890
\end{document}
```

Part VII
Fonts and features with XeTeX

21 XeTeX-only font features

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

21.1 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.

21.2 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.[12] 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 16.9 on page 48 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.

21.3 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.

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

Example 41: XeTeX's `Mapping` feature.

“¡A small amount of—text!”

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

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

22 Mac OS X's AAT fonts

> ***Warning!*** *XƎTEX'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 XƎTEX'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 XƎTEX (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'.

22.1 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 XƎTEX; 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').

22.2 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`.

22.3 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 10 on page 22.

22.4 Contextuals

This feature refers to glyph substitution that vary by their position; things like contextual swashes are implemented here. The options for AAT fonts are `WordInitial`, `WordFinal` (Example 42), `LineInitial`, `LineFinal`, and `Inner` (Example 43, also called 'non-final' sometimes). As non-exclusive selectors, like the ligatures, you can turn them off by prefixing their name with `No`.

22.5 Vertical position

The `VerticalPosition` feature is used to access things like subscript (`Inferior`) and superscript (`Superior`) numbers and letters (and a small amount of punctuation, sometimes). The `Ordinal` option is (supposed to be) contextually sensitive to only raise characters that appear directly after a number. These are shown in Example 44.

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

22.6 Fractions

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

In AAT fonts, the 'fraction slash' or solidus character, is to be used to create fractions. When `Fractions` are turned `On`, then only pre-drawn fractions will be used. See Example 45.

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

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

22.7 Variants

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

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

where is all the vegemite

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

Example 43: 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.
```

Example 44: Vertical position for AAT fonts.

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

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

Example 45: 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.

$\frac{1}{2}$ 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 46: Alternate design of pre-composed fractions.

1/2 1/4 5/6 13579/24680
$\frac{1}{2}$ $\frac{1}{4}$ $\frac{5}{6}$ 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
```

Example 47: 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}
```

See Section 23 on page 65 for a way to assign names to variants, which should be done on a per-font basis.

22.8 Alternates

Selection of `Alternates` again must be done numerically; see Example 48. See Section 23 on page 65 for a way to assign names to alternates, which should be done on a per-font basis.

22.9 Style

The options of the `Style` feature are defined in AAT as one of the following: `Display`, `Engraved`, `IlluminatedCaps`, `Italic`, `Ruby`,[13] `TallCaps`, or `TitlingCaps`.

Typical examples for these features are shown in Section 16.3.

22.10 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.

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

Example 48: 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}
```

22.11 Character width

See Section 16.6 on page 43 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.

22.12 Vertical typesetting

XƎTEX provides for vertical typesetting simply with the ability to rotate the individual glyphs as a font is used for typesetting, as shown in Example 49.

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

22.13 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.

22.14 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`.

Example 49: 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
```

Part VIII
Customisation and programming interface

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

23 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 XƎTEX file `AAT-info.tex` to obtain the code numbers. See Example 50.

`\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}
```

`\newfontfeature`

In case the above commands do not accommodate the desired font feature (perhaps a new XƎTEX 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 used to contain an AAT feature 'Avoid d-collisions'. To access it with this package, you could do some like the following:

```
\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
```

The advantage to using the `\newAATfeature` and `\newopentypefeature` commands instead of `\newfontfeature` is that they check if the selected font actually

Example 50: 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.
```

contains the desired font feature at load time. By contrast, `\newfontfeature` will not give a warning for improper input.

24 Defining new scripts and languages

`\newfontscript`
`\newfontlanguage`

While the scripts and languages listed in Table 16 and Table 17 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 23 on the preceding page) is the error-checking that is performed when the script or language is requested.

25 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 font feature selection using a literal feature selection string if 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]
```

26 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 52.

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

Example 51: Using raw font features directly.

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

Example 52: Renaming font features.

Roman Letters *And Swash*

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

Example 53: Renaming font feature options.

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

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

with the command `\aliasfontfeatureoption{`⟨*font feature*⟩`}{`⟨*existing name*⟩`}{`⟨*new name*⟩`}`, such as shown in Example 53.

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.

27 Programming interface

27.1 Variables

`\l_fontspec_family_tl`
`\l_fontspec_font`

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 from the 'base' font in the family is stored in `\l_fontspec_font`.

`\g_fontspec_encoding_tl`

Package authors who need to load fonts with legacy LaTeX NFSS commands may also need to know what the default font encoding is. Since this has changed from `EU1`/`EU2` to `TU`, it is best to use the variables `\g_fontspec_encoding_tl` or `\UTFencname` instead.

27.2 Functions for loading new fonts and families

`\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*⟩.

27.3 Conditionals

The following functions in expl3 syntax may be used for writing code that interfaces with fontspec-loaded fonts. The following conditionals are all provided in `TF`, `T`, and `F` forms.

27.3.1 Querying font families

`\fontspec_font_if_exist:nTF` Test whether the 'font name' (`#1`) exists or is loadable. The syntax of `#1` is a restricted/simplified version of fontspec's usual font loading syntax; fonts to be loaded by filename are detected by the presence of an appropriate extension (`.otf`, etc.), and paths should be included inline. E.g.:

```
\fontspec_font_if_exist:nTF {cmr10}{T}{F}
\fontspec_font_if_exist:nTF {Times~ New~ Roman}{T}{F}
\fontspec_font_if_exist:nTF {texgyrepagella-regular.otf}{T}{F}
\fontspec_font_if_exist:nTF {/Users/will/Library/Fonts/CODE2000.TTF}{T}{F}
```

The synonym `\IfFontExistsTF` is provided for 'document authors'.

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

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

27.3.2 Availability of features

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

`\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:nnnTF {`
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.

27.3.3 Currently selected features

`ntspec_if_current_feature:nTF` Test whether the currently loaded font is using the specified raw OpenType feature tag `#1`. The tag string `#1` should be prefixed with + to query an active feature, and with a - (hyphen) to query a disabled feature.

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

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

Part IX
Implementation

28 Loading

The expl3 module is `fontspec`.

```
⟨@@=fontspec⟩
```

Check engine and load specific modules. For LuaTeX, load luaotfload.

```
⟨*load⟩
\sys_if_engine_luatex:T
  { \RequirePackage{luaotfload}
    \directlua{require("fontspec")}
    \RequirePackageWithOptions{fontspec-luatex} \endinput }
\sys_if_engine_xetex:T
  { \RequirePackageWithOptions{fontspec-xetex}  \endinput }
```

If not one of the above, error:

```
\msg_new:nnn {fontspec} {cannot-use-pdftex}
 {
  The~ fontspec~ package~ requires~ either~ XeTeX~ or~ LuaTeX.\\\\
  You~ must~ change~ your~ typesetting~ engine~ to,~ e.g.,~ "xelatex"~ or~ "lualatex" instead~ of~ plain~ "latex"~ or~ "pdflatex".
 }
\msg_fatal:nn {fontspec} {cannot-use-pdftex}
\endinput
⟨/load⟩
```

29 Declaration of variables and functions

```
⟨*fontspec⟩
```

Booleans

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.

```
\bool_new:N \l_@@_firsttime_bool
```

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

These strange set functions are to simplify returning code from LuaTeX:

```
\bool_new:N \l_@@_check_bool
\cs_new:Npn \FontspecSetCheckBoolTrue  { \bool_set_true:N  \l_@@_check_bool }
\cs_new:Npn \FontspecSetCheckBoolFalse { \bool_set_false:N \l_@@_check_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
\bool_new:N \g_@@_euenc_bool
```

```
\bool_new:N \l_@@_disable_defaults_bool
\bool_new:N \l_@@_alias_bool
\bool_new:N \l_@@_external_bool
\bool_new:N \l_@@_never_check_bool
\bool_new:N \l_@@_defining_encoding_bool
\bool_new:N \l_@@_script_exist_bool
```

Counters

```
\int_new:N \l_@@_script_int
\int_new:N \l_@@_language_int
\int_new:N \l_@@_strnum_int
\int_new:N \l_@@_tmp_int
\int_new:N \l_@@_em_int
\int_new:N \l_@@_emdef_int
```

Floating point

```
\fp_new:N \l_@@_tmpa_fp
\fp_new:N \l_@@_tmpb_fp
```

Dimensions

```
\dim_new:N \l_@@_tmpa_dim
\dim_new:N \l_@@_tmpb_dim
\dim_new:N \l_@@_tmpc_dim
\seq_new:N \g_@@_bf_series_seq
```

Comma lists

```
\clist_new:N \g_@@_default_fontopts_clist
\clist_new:N \g_@@_all_keyval_modules_clist
\clist_set:Nn \l_@@_sizefeat_clist {Size={-}}
```

Property lists

```
\prop_new:N \g_@@_fontopts_prop
\prop_new:N \l_@@_nfss_prop
\prop_new:N \l_@@_nfssfont_prop
```

```
\prop_new:N \g_@@_OT_features_prop
\prop_new:N \g_@@_all_opentype_feature_names_prop
\prop_new:N \g_@@_em_prop
```

Token lists

```
\tl_new:N \g_@@_mathrm_tl
\tl_new:N \g_@@_bfmathrm_tl
\tl_new:N \g_@@_mathsf_tl
\tl_new:N \g_@@_mathtt_tl

\tl_new:N \l_@@_family_label_tl
\tl_new:N \l_@@_fake_slant_tl
\tl_new:N \l_@@_fake_embolden_tl

\tl_new:N \l_@@_fontname_up_tl
\tl_new:N \l_@@_fontname_bf_tl
\tl_new:N \l_@@_fontname_it_tl
\tl_new:N \l_@@_fontname_bfit_tl
\tl_new:N \l_@@_fontname_sl_tl
\tl_new:N \l_@@_fontname_bfsl_tl
\tl_new:N \l_@@_fontname_sc_tl

\tl_new:N \l_@@_fontfeat_up_clist
\tl_new:N \l_@@_fontfeat_bf_clist
\tl_new:N \l_@@_fontfeat_it_clist
\tl_new:N \l_@@_fontfeat_bfit_clist
\tl_new:N \l_@@_fontfeat_sl_clist
\tl_new:N \l_@@_fontfeat_bfsl_clist
\tl_new:N \l_@@_fontfeat_sc_clist

\tl_new:N \l_@@_script_name_tl
\tl_new:N \l_fontspec_script_tl
\tl_new:N \l_@@_lang_name_tl
\tl_new:N \l_fontspec_lang_tl

\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~}
\tl_set:Nn \g_@@_postadjust_tl { \l_@@_wordspace_adjust_tl \l_@@_punctspace_adjust_tl }
```

29.1 Generic functions

\@@_keys_set_known:nnN

```
\cs_new:Nn \@@_keys_set_known:nnN
  {
⟨debug⟩  \typeout{:::: Keys~set:~{#1}~{#2} }
    \keys_set_known:nnN {#1} {#2} #3
⟨debug⟩  \typeout{:::: Leftover:~{#3} }
  }
\cs_generate_variant:Nn \@@_keys_set_known:nnN {nx}
```

\@@_head_ii:n Expands to the first two ⟨*items*⟩ of #1.

```
\cs_set:Npn \@@_head_ii:n #1 { \@@_head_ii:w #1 *** \q_stop}
\cs_set:Npn \@@_head_ii:w #1#2#3 \q_stop { #1#2 }
\cs_generate_variant:Nn \@@_head_ii:n {o}
```

`\@@_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: }
  }
```

29.2 expl3 variants

```
\cs_generate_variant:Nn \int_set:Nn {Nv}
\cs_generate_variant:Nn \keys_set:nn {nx}
\cs_generate_variant:Nn \keys_set_known:nnN {nx}
\cs_generate_variant:Nn \prop_put:Nnn {Nxx}
\cs_generate_variant:Nn \prop_put:Nnn {NxV}
\cs_generate_variant:Nn \prop_gput:Nnn  {NxV}
\cs_generate_variant:Nn \prop_gput:Nnn  {Nxn}
\cs_generate_variant:Nn \prop_get:NnNT  {NxN}
\cs_generate_variant:Nn \prop_get:NnNTF {NxN}
\cs_generate_variant:Nn \str_if_eq:nnTF {nv}
\cs_generate_variant:Nn \tl_if_empty:nTF {x}
\cs_generate_variant:Nn \tl_if_empty:nF {x}
\cs_generate_variant:Nn \tl_if_empty:nF {f}
\cs_generate_variant:Nn \tl_if_eq:nnT {ox}
\cs_generate_variant:Nn \tl_replace_all:Nnn {Nnx}
```

⟨/fontspec⟩

30 Error/warning/info messages

⟨*fontspec⟩

Shorthands for messages:

```
\cs_new:Npn \@@_error:n     { \msg_error:nn     {fontspec} }
\cs_new:Npn \@@_error:nn    { \msg_error:nnn    {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} }
```

Allow messages to be written with spaces acting as normal:

```
\cs_generate_variant:Nn \msg_new:nnn  {nnx}
\cs_generate_variant:Nn \msg_new:nnnn {nnxx}
\cs_new:Nn \@@_msg_new:nnn
  { \msg_new:nnx {#1} {#2} { \tl_trim_spaces:n {#3} } }
\cs_new:Nn \@@_msg_new:nnnn
  { \msg_new:nnxx {#1} {#2} { \tl_trim_spaces:n {#3} } { \tl_trim_spaces:n {#4} } }
```

```
\char_set_catcode_space:n {32}
```

30.1 Errors

```
\@@_msg_new:nnn {fontspec} {only-inside-encdef}
 {
  \exp_not:N#1can only be used in the second argument
  to \string\DeclareUnicodeEncoding.
 }
\@@_msg_new:nnn {fontspec} {only-import-tu}
 {
  The "\string\ImportEncoding" command can only take "TU" as an argument at this stage.
 }
\@@_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.
 }
\@@_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.
 }
\@@_msg_new:nnnn {fontspec} {not-in-addfontfeatures}
 {
  The "#1" font feature cannot be used in \string\addfontfeatures.
 }
 {
  This is due to how TeX loads fonts; such settings
  are global so adding them mid-document within a group causes
  confusion. You'll need to define multiple font families to achieve
  what you want.
 }
```

30.2 Warnings

```
\@@_msg_new:nnn {fontspec} {tu-clash}
 {
  I have found the tuenc.def encoding definition file but the TU encoding is not
  defined by the LaTeX2e kernel; attempting to correct but you really should update
  to the latest version of LaTeX2e.
 }
\@@_msg_new:nnn {fontspec} {tu-missing}
 {
  The TU encoding seems to be missing; please update to the latest version of LaTeX2e.
 }
\@@_msg_new:nnn {fontspec} {addfontfeatures-ignored}
 {
  \string\addfontfeature (s) ignored \msg_line_context:;
  it cannot be used with a font that wasn't selected by a fontspec command.\\
  \\
  The current font is "\use:c{font@name}".\\
  \int_compare:nTF { \clist_count:n {#1} = 1 }
    { The requested feature is "#1". }
    { The requested features are "#1". }
 }
\@@_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.
 }
\@@_msg_new:nnn {fontspec} {font-index-needs-ttc}
 {
  The "FontIndex" feature is only supported by TTC (TrueType Collection) fonts.\\
  Feature ignored.
 }
\@@_msg_new:nnn {fontspec} {feat-cannot-remove}
 {
  The "#1" feature cannot be deactivated. Request ignored.
 }
```

30.3 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 NFSS series/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" 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
 }
```

Reset 'space' behaviour:

```
\char_set_catcode_ignore:n {32}
```

⟨/fontspec⟩

31 Opening code

31.1 Package options

```
\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{euenc}{\bool_set_true:N  \g_@@_euenc_bool}
\DeclareOption{tuenc}{\bool_set_false:N \g_@@_euenc_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,tuenc}
\ProcessOptions*
```

31.2 Encodings

Soon to be the default, with a just-in-case check:

```
\bool_if:NF \g_@@_euenc_bool
```

```
  {
    \file_if_exist:nTF {tuenc.def}
      {
        \cs_if_exist:cF {T@TU}
          {
            \@@_warning:n {tu-clash}
            \DeclareFontEncoding{TU}{}{}
            \DeclareFontSubstitution{TU}{lmr}{m}{n}
          }
      }
      {
        \@@_warning:n {tu-missing}
        \bool_set_true:N \g_@@_euenc_bool
      }
  }
\bool_if:NTF \g_@@_euenc_bool
  {
⟨xetexx⟩    \tl_set:Nn \g_fontspec_encoding_tl {EU1}
⟨luatex⟩    \tl_set:Nn \g_fontspec_encoding_tl {EU2}
  }
  { \tl_set:Nn \g_fontspec_encoding_tl { TU } }
```

```
\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 if needed
```

To overcome the encoding changing the current font size, but only if a class has been loaded first:

```
\tl_if_in:NnT \@filelist {.cls} { \normalsize }
```

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.

```
\bool_if:NT \g_@@_euenc_bool
  {
⟨luatex⟩    \cs_set_eq:NN \fontspec_tmp: \XeTeXpicfile
⟨luatex⟩    \cs_set:Npn \XeTeXpicfile {}
    \RequirePackage{xunicode}
⟨luatex⟩    \cs_set_eq:NN \XeTeXpicfile \fontspec_tmp:
  }
```

32 expl3 interface for font loading

n,\@@_primitive_font_gset:Nnn

```
\cs_set:Npn \@@_primitive_font_set:Nnn #1#2#3
  {
    \font #1 = #2 ~at~ #3 \scan_stop:
  }
\cs_set:Npn \@@_primitive_font_gset:Nnn #1#2#3
  {
    \global \font #1 = #2 ~at~ #3 \scan_stop:
  }
```

ont_suppress_not_found_error:

```
\cs_set:Npn \@@_font_suppress_not_found_error:
  {
    \int_set_eq:NN \xetex_suppressfontnotfounderror:D \c_one
  }
```

[pTF]@_primitive_font_if_null:N

```
\prg_set_conditional:Nnn \@@_primitive_font_if_null:N {p,TF,T,F}
  {
    \ifx #1 \nullfont
      \prg_return_true:
    \else
      \prg_return_false:
    \fi
  }
```

[TF]@_primitive_font_if_exist:n

```
\prg_set_conditional:Nnn \@@_primitive_font_if_exist:n {TF,T,F}
  {
    \group_begin:
      \@@_font_suppress_not_found_error:
      \@@_primitive_font_set:Nnn \l_@@_primitive_font {#1} {10pt}
      \@@_primitive_font_if_null:NTF \l_@@_primitive_font
        { \group_end: \prg_return_false: }
        { \group_end: \prg_return_true:  }
  }
```

tive_font_glyph_if_exist:NnTF

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

33 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 35.1 on page 93.

33.0.1 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 O{} }
 {
  \fontspec_set_family:Nnn \f@family {#1,#3} {#2}
  \fontencoding { \l_@@_nfss_enc_tl }
  \selectfont
  \ignorespaces
 }
```

\setmainfont

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 O{} }
 {
  \fontspec_set_family:Nnn \g_@@_rmfamily_family {#1,#3} {#2}
  \tl_set_eq:NN \rmdefault \g_@@_rmfamily_family
  \use:x { \exp_not:n { \DeclareRobustCommand \rmfamily }
   {
    \exp_not:N \fontencoding { \l_@@_nfss_enc_tl }
    \exp_not:N \fontfamily { \g_@@_rmfamily_family }
    \exp_not:N \selectfont
   }
  }
  \str_if_eq_x:nnT {\familydefault} {\rmdefault}
   { \tl_set_eq:NN \encodingdefault \l_@@_nfss_enc_tl }
  \normalfont
  \ignorespaces
 }
```

\setsansfont

```
\DeclareDocumentCommand \setsansfont { O{} m O{} }
 {
  \fontspec_set_family:Nnn \g_@@_sffamily_family {#1,#3} {#2}
  \tl_set_eq:NN \sfdefault \g_@@_sffamily_family
  \use:x { \exp_not:n { \DeclareRobustCommand \sffamily }
   {
    \exp_not:N \fontencoding { \l_@@_nfss_enc_tl }
    \exp_not:N \fontfamily { \g_@@_sffamily_family }
    \exp_not:N \selectfont
   }
  }
  \str_if_eq_x:nnT {\familydefault} {\sfdefault}
    { \tl_set_eq:NN \encodingdefault \l_@@_nfss_enc_tl }
  \normalfont
  \ignorespaces
 }
```

\setmonofont

```
\DeclareDocumentCommand \setmonofont { O{} m O{} }
 {
  \fontspec_set_family:Nnn \g_@@_ttfamily_family {#1,#3} {#2}
  \tl_set_eq:NN \ttdefault \g_@@_ttfamily_family
  \use:x { \exp_not:n { \DeclareRobustCommand \ttfamily }
   {
    \exp_not:N \fontencoding { \l_@@_nfss_enc_tl }
    \exp_not:N \fontfamily { \g_@@_ttfamily_family }
    \exp_not:N \selectfont
   }
  }
  \str_if_eq_x:nnT {\familydefault} {\ttdefault}
    { \tl_set_eq:NN \encodingdefault \l_@@_nfss_enc_tl }
  \normalfont
  \ignorespaces
 }
```

\setromanfont

This is the old name for `\setmainfont`, retained ad infinitum for backwards compatibility. It was deprecated in 2010.

```
\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`.

```
\DeclareDocumentCommand \setmathrm { O{} m O{} }
 {
  \fontspec_set_family:Nnn \g_@@_mathrm_tl {#1} {#2}
 }

\DeclareDocumentCommand \setboldmathrm { O{} m O{} }
 {
  \fontspec_set_family:Nnn \g_@@_bfmathrm_tl {#1} {#2}
 }

\DeclareDocumentCommand \setmathsf { O{} m O{} }
 {
  \fontspec_set_family:Nnn \g_@@_mathsf_tl {#1} {#2}
 }

\DeclareDocumentCommand \setmathtt { O{} m O{} }
 {
  \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`

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 O{} }
 {
  \fontspec_set_family:cnn { g_@@_ \cs_to_str:N #1 _family } {#2,#4} {#3}
  \use:x
   {
    \exp_not:N \DeclareRobustCommand \exp_not:N #1
     {
      \exp_not:N \fontfamily { \use:c {g_@@_ \cs_to_str:N #1 _family} }
      \exp_not:N \fontencoding { \l_@@_nfss_enc_tl }
      \exp_not:N \selectfont
     }
   }
 }
```

`\newfontface`

`\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 O{} }
 {
  \newfontfamily #1 [ BoldFont={},ItalicFont={},SmallCapsFont={},#2,#4 ] {#3}
 }
```

33.0.2 Font feature selection

`\defaultfontfeatures`

This macro takes one argument that consists of all of feature options that will be applied by default to all subsequent `\fontspec`, et al., commands. It stores its value in `\g_fontspec_default_fontopts_tl` (initialised empty), which is concatenated with the individual macro choices in the [...] macro.

```
\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,} }
     }
   }
 }
```

\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 \@@_select_font_family:nn), so this means that the only added features to the font are strictly those specified by this command.

\addfontfeature is defined as an alias, as I found that I often typed this instead when adding only a single font feature.

```
\DeclareDocumentCommand \addfontfeatures {m}
 {
⟨debug⟩  \typeout{^^J::::::::::::::::::::::::::::::::::^^J: addfontfeatures}
  \fontspec_if_fontspec_font:TF
   {
    \group_begin:
      \keys_set_known:nnN {fontspec-addfeatures} {#1} \l_@@_tmp_tl
      \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
⟨debug⟩           \typeout{ \@@_select_font_family:nn { \l_@@_options_tl , #1 } {\l_@@_fontname
      \use:x
       {
        \@@_select_font_family:nn
          { \l_@@_options_tl , #1 } {\l_@@_fontname_tl}
       }
    \group_end:
    \fontfamily\l_fontspec_family_tl\selectfont
   }
   {
    \@@_warning:nx {addfontfeatures-ignored} {#1}
   }
   \ignorespaces
 }
\cs_set_eq:NN \addfontfeature \addfontfeatures
```

33.0.3 Defining new font features

\newfontfeature

\newfontfeature takes two arguments: the name of the feature tag by which to reference it, and the string that is used to select the font feature.

```
\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_aat_feature_group:n {#1} }
  \keys_if_choice_exist:nnnT {fontspec} {#1} {#2}
    { \@@_warning:nxx {feature-option-overwrite} {#1} {#2} }
  \@@_define_aat_feature:nnnn {#1}{#2}{#3}{#4}
 }
```

\newopentypefeature

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

```
\DeclareDocumentCommand \newopentypefeature {mmm}
 {
  \keys_if_exist:nnF { fontspec / options } {#1}
    { \@@_define_opentype_feature_group:n {#1} }
  \keys_if_choice_exist:nnnT {fontspec} {#1} {#2}
    { \@@_warning:nxx {feature-option-overwrite} {#1} {#2} }
  \@@_define_opentype_feature:nnn {#1}{#2}{#3}
 }
```

\newICUfeature

```
\cs_set_eq:NN \newICUfeature \newopentypefeature % deprecated
```

\aliasfontfeature User commands for renaming font features and font feature options.

```
\DeclareDocumentCommand \aliasfontfeature {mm}
 {
  \bool_set_false:N \l_@@_alias_bool

  \clist_map_inline:nn
   { fontspec, fontspec-opentype, fontspec-aat, fontspec-preparse, fontspec-preparse-external,
     fontspec-preparse-nested, fontspec-renderer }
   {

     \keys_if_exist:nnT {##1} {#1}
      {
        \bool_set_true:N \l_@@_alias_bool
        \keys_define:nn {##1}
          { #2 .code:n = { \keys_set:nn {##1} { #1 = {####1} } } }
      }
   }

  \bool_if:NF \l_@@_alias_bool
    { \@@_warning:nx {rename-feature-not-exist} {#1} }
 }
```

\aliasfontfeatureoption

```
\DeclareDocumentCommand \aliasfontfeatureoption {mmm}
 {
  \bool_set_false:N \l_@@_alias_bool

  \clist_map_inline:Nn \g_@@_all_keyval_modules_clist
   {

     \keys_if_exist:nnT {##1} {#1}
      {
        \bool_set_true:N \l_@@_alias_bool
        \keys_define:nn { ##1 / #1 } { #3 .meta:n = {#2} }
      }
   }

  \bool_if:NF \l_@@_alias_bool
```

```
    { \@@_warning:nx {rename-feature-not-exist} {#1} }
 }
```

\newfontscript

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

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

\newfontlanguage

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

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

\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}
```

\IfFontFeaturesTF

```
\DeclareDocumentCommand \IfFontFeatureActiveTF {mmm}
  {
⟨debug⟩     \typeout{^^J::::::::::::::::::::::::::::::::::::::::::::::}
⟨debug⟩     \typeout{:IfFontFeatureActiveTF \exp_not:n{{#1}{#2}{#3}}}
    \@@_if_font_feature:nTF {#1} {#2} {#3}
  }
\prg_new_conditional:Nnn \@@_if_font_feature:n {TF}
  {
    \tl_gclear:N \g_@@_single_feat_tl
    \group_begin:
      \@@_font_suppress_not_found_error:
      \@@_init:
      \bool_set_true:N \l_@@_ot_bool
      \bool_set_true:N \l_@@_never_check_bool
      \bool_set_false:N \l_@@_firsttime_bool
      \clist_clear:N \l_@@_fontfeat_clist
      \@@_get_features:Nn \l_@@_rawfeatures_sclist {#1}
    \group_end:

⟨debug⟩    \typeout{:::> \exp_not:N\l_@@_rawfeatures_sclist->~{\l_@@_rawfeatures_sclist}}
⟨debug⟩    \typeout{:::> \exp_not:N\g_@@_single_feat_tl->~{\g_@@_single_feat_tl}}

    \tl_if_empty:NTF \g_@@_single_feat_tl { \prg_return_false: }
      {
        \exp_args:NV \fontspec_if_current_feature:nTF \g_@@_single_feat_tl
```

```
          { \prg_return_true: } { \prg_return_false: }
      }
  }
```

34 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
    \@@_primitive_font_set:Nnn \l_fontspec_font {\l_@@_fontdef_tl} {\f@size pt}
    \bool_if:NTF \l_@@_atsui_bool
     {
      \@@_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
   \@@_primitive_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
   \@@_primitive_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_@@_script_int {\l_@@_tmp_tl}

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

     \@@_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 {la
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
   \@@_primitive_font_set:Nnn \l_fontspec_font {\l_@@_fontdef_tl} {\f@size pt}
   \@@_set_font_type:
     \bool_if:NTF \l_@@_ot_bool
     {
      \@@_iv_str_to_num:Nn \l_@@_script_int {#1}
      \@@_iv_str_to_num:Nn \l_@@_language_int {#2}
      \@@_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
    \@@_primitive_font_set:Nnn \l_fontspec_font {\l_@@_fontdef_tl} {\f@size pt}
    \@@_set_font_type:
    \bool_if:NTF \l_@@_ot_bool
     {
      \@@_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
    \@@_primitive_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_@@_script_int {\l_@@_tmp_tl}
      \prop_get:cnN {g_@@_ \f@family _prop} {script-tag}  \l_fontspec_script_tl

      \@@_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
    \@@_primitive_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}
      \@@_iv_str_to_num:Nn \l_@@_script_int {#1}
      \@@_check_lang:nTF {#2} \prg_return_true: \prg_return_false:
     }
     { \prg_return_false: }
   }
   { \prg_return_false: }
 }
```

ontspec_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
    \@@_primitive_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: }
 }
```

tspec_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
    \@@_primitive_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 `\@@_select_font_family:nn`, which may change in the future.

```
\cs_new:Nn \fontspec_set_family:Nnn
 {
  \tl_set:Nn \l_@@_family_label_tl { #1 }
  \@@_select_font_family: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 }
  \@@_select_font_family:nn {#3}{#4}
  \tl_set_eq:NN #1 \l_fontspec_font
  \tl_set_eq:NN #2 \l_fontspec_family_tl
 }
```

\fontspec_font_if_exist:n

```
\prg_new_conditional:Nnn \fontspec_font_if_exist:n {TF,T,F}
  {
    \group_begin:
      \@@_init:
      \@@_if_detect_external:nT {#1} { \@@_font_is_file: }
      \@@_primitive_font_if_exist:nTF { \@@_construct_font_call:nn {#1} {} }
        { \group_end: \prg_return_true: }
        { \group_end: \prg_return_false:  }
  }
\cs_set_eq:NN \IfFontExistsTF \fontspec_font_if_exist:nTF
```

ntspec_if_current_feature:nTF

Test whether the currently loaded font is using the specified raw OpenType feature tag #1.

```
\prg_new_conditional:Nnn \fontspec_if_current_feature:n {TF,T,F}
 {
   \exp_args:Nxx \tl_if_in:nnTF
     { \fontname\font } { \tl_to_str:n {#1} }
     { \prg_return_true: } { \prg_return_false: }%
 }
```

35 Internals

35.1 The main function for setting fonts

\@@_select_font_family: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_@@_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_new_protected:Nn \@@_select_font_family:nn
 {
⟨debug⟩\typeout{^^J^^J::::::::::::::::::::::::::::::^^J:: fontspec_select:nn~ {#1}~ {#2} }
  \group_begin:
  \@@_font_suppress_not_found_error:
  \@@_init:

  \@@_sanitise_fontname:Nn \l_fontspec_fontname_tl    {#2}
  \@@_sanitise_fontname:Nn \l_@@_fontname_up_tl {#2}
  \@@_sanitise_fontname:Nn \l_@@_basename_tl          {#2}

  \@@_if_detect_external:nT {#2}
   { \keys_set:nn {fontspec-preparse-external} {Path} }

  \@@_init_ttc:n {#2}
  \@@_load_external_fontoptions:Nn \l_fontspec_fontname_tl {#2}

  \@@_extract_all_features:n {#1}
  \tl_set:Nx \l_@@_fontid_tl { \tl_to_str:N \l_fontspec_fontname_tl-:-\tl_to_str:N \l_@@_all_f

⟨debug⟩\typeout{fontid: \l_@@_fontid_tl}

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

  \@@_save_family_needed:nTF {#2}
   {
     \@@_save_family:nn {#1} {#2}
⟨debug⟩  \@@_warning:nxx {defining-font} {#1} {#2}
   }
   {
⟨debug⟩  \typeout{Font~ family~ already~ defined.}
   }
  \group_end:
 }
```

\fontspec_select:nn

This old name has been used by 3rd party packages so for compatibility:

```
\cs_set_eq:NN \fontspec_select:nn \@@_select_font_family:nn
```

\@@_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_if_in:NnT #1 {##1}
       {
         \tl_remove_once:Nn #1 {##1}
         \tl_set:Nn \l_@@_extension_tl {##1}
         \clist_map_break:
       }
   }
 }
```

\@@_if_detect_external:nT

Check if either the fontname ends with a known font extension.

```
\prg_new_conditional:Nnn \@@_if_detect_external:n {T}
 {
⟨debug⟩  \typeout{:: @@_if_detect_external:n  { \exp_not:n {#1} } }
  \clist_map_inline:Nn \l_@@_extensions_clist
   {
    \bool_set_false:N \l_@@_tmpa_bool
    \exp_args:Nx % <- this should be handled earlier
    \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:
 }
```

\@@_init_ttc:n

For TTC fonts we assume they will be loading the italic/bold fonts from the same file, so prepopulate the fontnames to avoid needing to do it manually.

```
\cs_new:Nn \@@_init_ttc:n
 {
  \str_if_eq_x:nnT { \str_lower_case:f {\l_@@_extension_tl} } {.ttc}
   {
    \@@_sanitise_fontname:Nn \l_@@_fontname_it_tl   {#1}
    \@@_sanitise_fontname:Nn \l_@@_fontname_bf_tl   {#1}
    \@@_sanitise_fontname:Nn \l_@@_fontname_bfit_tl {#1}
   }
 }
```

_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
 {
⟨debug⟩  \typeout{:: @@_load_external_fontoptions:Nn \exp_not:N #1 {#2} }
  \@@_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_all_features:

```
\cs_new:Nn \@@_extract_all_features:n
 {
⟨debug⟩  \typeout{:: @@_extract_all_features:n { \unexpanded {#1} } }
  \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
     }
   }
 }
```

\@@_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:
 {
⟨debug⟩  \typeout{:: @@_preparse_features:}
```

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

```

  \@@_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_@@_fontname_up_tl), this latter is the new 'general font name' to use.

```
  \tl_set_eq:NN \l_fontspec_fontname_tl \l_@@_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:
 {
⟨debug⟩\typeout{:: @@_load_font}
⟨debug⟩\typeout{Set~ base~ font~ for~ preliminary~ analysis: \@@_construct_font_call:nn { \l_@
  \@@_primitive_font_set:Nnn \l_fontspec_font
     { \@@_construct_font_call:nn { \l_@@_fontname_up_tl } {} } {\f@size pt}
  \@@_primitive_font_if_null:NT \l_fontspec_font { \@@_error:nx {font-not-found} {\l_@@_fontname_up_tl} }
  \@@_set_font_type:
⟨debug⟩\typeout{Set~ base~ font~ properly: \@@_construct_font_call:nn { \l_@@_fontname_up_tl }
  \@@_primitive_font_gset:Nnn \l_fontspec_font
     { \@@_construct_font_call:nn { \l_@@_fontname_up_tl } {} } {\f@size pt}
  \l_fontspec_font % this is necessary for LuaLaTeX to check the scripts properly
 }
```

\@@_construct_font_call:nn

Constructs the complete font invocation. #1 : Base name
#2 : Extension
#3 : TTC Index
#4 : Renderer
#5 : Optical size
#6 : Font features

We check if ⟨*Font features*⟩ are empty and if so don't add in the separator colon.

```
\cs_set:Nn \@@_construct_font_call:nnnnnn
 {
  " \@@_fontname_wrap:n { #1 #2 #3 } #4 #5
    \str_if_eq_x:nnF {#6}{} {:#6} "
 }
```

In practice, we don't use the six-argument version, since most arguments are constructed on-the-fly:

```
\cs_set:Nn \@@_construct_font_call:nn
 {
  \@@_construct_font_call:nnnnnn
    {#1}
    \l_@@_extension_tl
    \l_@@_ttc_index_tl
    \l_fontspec_renderer_tl
    \l_@@_optical_size_tl
    {#2}
 }
```

nt_is_file:,\@@_font_is_name:

The \@@_fontname_wrap:n command takes the font name and either passes it through unchanged or wraps it in the syntax for loading a font 'by filename'. XƎTEX's syntax is followed since luaotfload provides compatibility.

```
\cs_new:Nn \@@_font_is_name:
  {
    \cs_set_eq:NN \@@_fontname_wrap:n \use:n
  }
\cs_new:Nn \@@_font_is_file:
  {
⟨xetexx⟩     \cs_set:Npn \@@_fontname_wrap:n ##1 { [ \l_@@_font_path_tl ##1 ] }
⟨luatex⟩     \cs_set:Npn \@@_fontname_wrap:n ##1 { file:\l_@@_font_path_tl ##1 }
  }
```

\@@_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
     {
      \@@_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-opentype} {Script=\l_@@_script_name_tl}
        \keys_set:nx {fontspec-opentype} {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-opentype} {Script=\l_@@_script_name_tl}
      \keys_set:nx {fontspec-opentype} {Language=\l_@@_lang_name_tl}
     }
   }
 }
```

\@@_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.

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

```
\cs_set:Nn \@@_get_features:Nn
 {
⟨debug⟩  \typeout{:: @@_get_features:Nn \exp_not:N #1 { \exp_not:n {#2} } }
  \@@_init_fontface:
  \@@_keys_set_known:nxN {fontspec-renderer} {\l_@@_fontfeat_clist,#2}
    \l_@@_keys_leftover_clist
  \@@_keys_set_known:nxN {fontspec} {\l_@@_keys_leftover_clist} \l_@@_keys_leftover_clist
⟨*xetexx⟩
  \bool_if:NTF \l_@@_ot_bool
    {
⟨debug⟩  \typeout{::: Setting~ keys~ for~ OpenType~ font~ features:~"\l_@@_keys_leftover_clist
      \keys_set:nx {fontspec-opentype} {\l_@@_keys_leftover_clist}
    }
    {
⟨debug⟩  \typeout{::: Setting~ keys~ for~ AAT~ font~ features:~"\l_@@_keys_leftover_clist"}
      \bool_if:NT \l_@@_atsui_bool
        { \keys_set:nx {fontspec-aat} {\l_@@_keys_leftover_clist} }
    }
⟨/xetexx⟩
⟨*luatex⟩
```

```
⟨debug⟩  \typeout{::: Setting~ keys~ for~ OpenType~ font~ features:~"\l_@@_keys_leftover_clist
  \keys_set:nx {fontspec-opentype} {\l_@@_keys_leftover_clist}
⟨/luatex⟩

  \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
 }
```

\@@_save_family_needed: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_needed:n {TF}
 {

⟨debug⟩  \typeout{save~ family:~ #1}
⟨debug⟩  \typeout{== fontid_tl: "\l_@@_fontid_tl".}

  \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 spaces zapped:
    \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_family:nn

Saves the relevant font information for future processing.

```
\cs_new:Nn \@@_save_family:nn
  {
    \@@_save_fontinfo:n {#2}
    \@@_find_autofonts:
    \DeclareFontFamily{\l_@@_nfss_enc_tl}{\l_fontspec_family_tl}{}
    \@@_set_faces:
    \@@_info:nxx {defining-font} {#1} {#2}
  }
```

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

```
\cs_new:Nn \@@_save_fontinfo:n
 {
  \prop_new:c {g_@@_ \l_fontspec_family_tl _prop}
  \prop_gput:cnx {g_@@_ \l_fontspec_family_tl _prop} {fontname} { #1 }
  \prop_gput:cnx {g_@@_ \l_fontspec_family_tl _prop} {options}  { \l_@@_all_features_clist }
  \prop_gput:cnx {g_@@_ \l_fontspec_family_tl _prop} {fontdef}
   {
    \@@_construct_font_call:nn {\l_fontspec_fontname_tl}
      { \l_@@_pre_feat_sclist \l_@@_rawfeatures_sclist }
   }
  \prop_gput:cnV {g_@@_ \l_fontspec_family_tl _prop} {script-num} \l_@@_script_int
  \prop_gput:cnV {g_@@_ \l_fontspec_family_tl _prop} {lang-num} \l_@@_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
 }
```

35.2 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_@@_fontname_bfit_tl {\l_@@_fontname_it_tl} {/B}
    \@@_set_autofont:Nnn \l_@@_fontname_bfit_tl {\l_@@_fontname_bf_tl} {/I}
    \@@_set_autofont:Nnn \l_@@_fontname_bfit_tl {\l_fontspec_fontname_tl} {/BI}
   }

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

  \bool_if:NF \l_@@_noit_bool
   {
```

```
   \@@_set_autofont:Nnn \l_@@_fontname_it_tl {\l_fontspec_fontname_tl} {/I}
  }

 \@@_set_autofont:Nnn \l_@@_fontname_bfsl_tl {\l_@@_fontname_sl_tl} {/B}
}
```

\@@_set_faces:

```
\cs_new:Nn \@@_set_faces:
 {
  \@@_add_nfssfont:nnnn \mddefault \updefault \l_fontspec_fontname_tl      \l_@@_fontfeat_up_c
  \@@_add_nfssfont:nnnn \bfdefault \updefault \l_@@_fontname_bf_tl   \l_@@_fontfeat_bf_clist
  \@@_add_nfssfont:nnnn \mddefault \itdefault \l_@@_fontname_it_tl   \l_@@_fontfeat_it_clist
  \@@_add_nfssfont:nnnn \mddefault \sldefault \l_@@_fontname_sl_tl   \l_@@_fontfeat_sl_clist
  \@@_add_nfssfont:nnnn \bfdefault \itdefault \l_@@_fontname_bfit_tl \l_@@_fontfeat_bfit_clist
  \@@_add_nfssfont:nnnn \bfdefault \sldefault \l_@@_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}
 }
```

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 {~}
 }
```

\@@_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:nxN {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} }
   }
 }
```

35.2.1 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:
 {
⟨debug⟩  \typeout{:: @@_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 XƎTEX 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_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}
 {
  \@@_primitive_font_set:Nnn \l_tmpa_font { \@@_construct_font_call:nn {#1} {}  } {\f@size pt
  \@@_primitive_font_set:Nnn \l_tmpb_font { \@@_construct_font_call:nn {#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.

```
\cs_new:Nn \@@_make_font_shapes:Nnnnn
 {
  \group_begin:
    \@@_keys_set_known:nxN {fontspec-preparse-external} { #4 } \l_@@_leftover_clist
    \@@_load_fontname:n {#1}
    \@@_declare_shape:nnxx {#2} {#3} { \l_@@_fontopts_clist, \l_@@_leftover_clist } {#5}
  \group_end:
 }

\cs_new:Nn \@@_load_fontname:n
 {
⟨debug⟩     \typeout{:: @@_load_fontname:n {#1} }
```

```
    \@@_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 }
    \@@_primitive_font_set:Nnn \l_fontspec_font { \@@_construct_font_call:nn {\l_fontspec_font
    \@@_primitive_font_if_null:NT \l_fontspec_font { \@@_error:nx {font-not-found} {#1} }
 }
```

\@@_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
 {
⟨debug⟩\typeout{=~ declare_shape:~{\l_fontspec_fontname_tl}~{#1}~{#2}}
  \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} { \@@_setup_single_size:nn {#3} {##1} }

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

\@@_setup_single_size:nn

```
\cs_new:Nn \@@_setup_single_size:nn
  {
    \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 #2 }
      \l_@@_sizing_leftover_clist
    \tl_if_empty:NT \l_@@_size_tl { \@@_error:n {no-size-info} }
⟨debug⟩\typeout{==~ size:~\l_@@_size_tl}

    % "normal"
    \@@_load_fontname:n {\l_@@_sizedfont_tl}
    \@@_setup_nfss:Nnnn \l_@@_nfss_tl {#1} {\l_@@_sizing_leftover_clist} {}
⟨debug⟩    \typeout{===~ sized~ font:~ \l_@@_sizedfont_tl}

    % 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_@@_fontname_sc_tl
       {
        \@@_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_@@_fontname_sc_tl} }% local for each size
     }

    \bool_if:NF \l_@@_nosc_bool
     {
      \@@_setup_nfss:Nnnn \l_@@_nfss_sc_tl
        {#1} {\l_@@_sizing_leftover_clist} {\l_@@_fontfeat_curr_clist}
     }
  }
```

\@@_setup_nfss:Nnnn

```
\cs_new:Nn \@@_setup_nfss:Nnnn
 {
⟨debug⟩\typeout{====~Setup~NFSS~shape:~<\l_@@_size_tl>~\l_fontspec_fontname_tl}

  \@@_get_features:Nn \l_@@_rawfeatures_sclist { #2 , #3 , #4 }
⟨debug⟩\typeout{====~Gathered~features:~\l_@@_rawfeatures_sclist}

  \tl_put_right:Nx #1
   {
    <\l_@@_size_tl> \l_@@_scale_tl
      \@@_construct_font_call:nn { \l_fontspec_fontname_tl }
        { \l_@@_pre_feat_sclist \l_@@_rawfeatures_sclist }
   }
 }
```

\@@_declare_shapes_normal:nn

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

\@@_declare_shapes_smcaps:nn

```
\cs_new:Nn \@@_declare_shapes_smcaps:nn
  {
   \tl_if_empty:NF \l_@@_nfss_sc_tl
     {
      \@@_DeclareFontShape:xxxxxx {\l_@@_nfss_enc_tl} {\l_fontspec_family_tl} {#1}
        { \@@_combo_sc_shape:n {#2} } {\l_@@_nfss_sc_tl} {\l_@@_postadjust_tl}
     }
  }

\cs_new:Nn \@@_combo_sc_shape:n
  {
    \tl_if_exist:cTF { \@@_shape_merge:nn {#1} {\scdefault} }
         { \tl_use:c { \@@_shape_merge:nn {#1} {\scdefault} } }
         { \scdefault }
  }
```

\@@_DeclareFontShape:nnnnnn

```
\cs_new:Nn \@@_DeclareFontShape:nnnnnn
 {
⟨debug⟩\typeout{DeclareFontShape:~{#1}{#2}{#3}{#4}...}
  \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}
```

\@@_declare_shape_slanted:nn

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 {\l_@@_nfss_enc_tl}{\l_fontspec_family_tl}{#1}{\sldefault}
      {<->ssub*\l_fontspec_family_tl/#1/\itdefault}{\l_@@_postadjust_tl}
   }
 }
```

\@@_declare_shape_loginfo:nn

Lastly some informative messaging.

```
\cs_new:Nn \@@_declare_shape_loginfo:nn
 {
  \tl_gput_right:Nx \l_fontspec_defined_shapes_tl
   {
    \exp_not:n { \\ }
   -~ \exp_not:N \str_case:nn {#1/#2}
    {
      {\mddefault/\updefault} {'normal'~}
      {\bfdefault/\updefault} {'bold'~}
      {\mddefault/\itdefault} {'italic'~}
      {\mddefault/\sldefault} {'slanted'~}
      {\bfdefault/\itdefault} {'bold~ italic'~}
      {\bfdefault/\sldefault} {'bold~ slanted'~}
    } (#1/#2)~
   with~ NFSS~ spec.:~
   \l_@@_nfss_tl
   \exp_not:n { \\ }
   -~ \exp_not:N \str_case:nn { #1 / \@@_combo_sc_shape:n {#2} }
    {
      {\mddefault/\scdefault} {'small~ caps'~}
      {\bfdefault/\scdefault} {'bold~ small~ caps'~}
      {\mddefault/\itscdefault} {'italic~ small~ caps'~}
      {\bfdefault/\itscdefault} {'bold~ italic~ small~ caps'~}
      {\mddefault/\slscdefault} {'slanted~ small~ caps'~}
      {\bfdefault/\slscdefault} {'bold~ slanted~ small~ caps'~}
    }~( #1 / \@@_combo_sc_shape:n {#2} )~
   with~ NFSS~ spec.:~
   \l_@@_nfss_sc_tl
   \tl_if_empty:fF {\l_@@_postadjust_tl}
    {
     \exp_not:N \\ and~ font~ adjustment~ code: \exp_not:N \\ \l_@@_postadjust_tl
    }
  }
}
```

Maybe \str_if_eq_x:nnF would be better?

35.2.2 Features

\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⟩
```

\@@_make_ot_smallcaps:TF

This macro checks if the font contains small caps.

```
⟨luatex⟩\cs_set:Nn \@@_make_smallcaps:TF
⟨xetexx⟩\cs_set:Nn \@@_make_ot_smallcaps:TF
 {
  \@@_check_ot_feat:nTF {smcp} {#1} {#2}
 }
⟨*xetexx⟩
\cs_set:Nn \@@_make_smallcaps:TF
 {
   \bool_if:NTF \l_@@_ot_bool
   { \@@_make_ot_smallcaps:TF {#1} {#2} }
   {
     \bool_if:NT \l_@@_atsui_bool
      { \@@_make_AAT_feature_string:nnTF {3}{3} {#1} {#2} }
   }
 }
⟨/xetexx⟩
```

\@@_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
  {
⟨debug⟩          \typeout{:::: @@_update_featstr:n {#1}}
    \bool_if:NF \l_@@_firsttime_bool
      {
        \tl_gset:Nx \g_@@_single_feat_tl { #1 }
⟨debug⟩          \typeout{::::~ Adding~ feature.}
        \tl_gput_right:Nx  \l_@@_rawfeatures_sclist {#1;}
      }
  }
```

\@@_remove_clashing_featstr:n

```
\cs_new:Nn \@@_remove_clashing_featstr:n
  {
⟨debug⟩    \typeout{:::: @@_remove_clashing_featstr:n {#1}}
    \clist_map_inline:nn {#1}
      {
⟨debug⟩        \typeout{::::~ Removing~ feature~ "##1;"}
        \tl_gremove_all:Nn \l_@@_rawfeatures_sclist {##1;}
      }
  }
```

35.3 Initialisation

\@@_init: Initialisations that need to occur once per fontspec font invocation. (Some of these may be redundant. Check whether they're assigned to globally or not.)

```
\cs_set:Npn \@@_init:
 {
⟨debug⟩  \typeout{:: @@_init:}
  \bool_set_false:N \l_@@_ot_bool
  \bool_set_true:N \l_@@_firsttime_bool
  \@@_font_is_name:
  \tl_clear:N \l_@@_font_path_tl
  \tl_clear:N \l_@@_optical_size_tl
  \tl_clear:N \l_@@_ttc_index_tl
  \tl_clear:N \l_fontspec_renderer_tl
  \tl_clear:N \l_fontspec_defined_shapes_tl
  \tl_clear:N \g_@@_curr_series_tl
  \tl_gset_eq:NN \l_@@_nfss_enc_tl \g_fontspec_encoding_tl

⟨*luatex⟩
  \tl_set:Nn \l_fontspec_mode_tl {node}
  \int_set:Nn \luatex_prehyphenchar:D { `\- } % fixme
  \int_zero:N \luatex_posthyphenchar:D        % fixme
  \int_zero:N \luatex_preexhyphenchar:D       % fixme
  \int_zero:N \luatex_postexhyphenchar:D      % fixme
⟨/luatex⟩
 }
```

\@@_init_fontface: Executed in \@@_get_features:Nn.

```
\cs_new:Nn \@@_init_fontface:
  {
    \tl_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
  }
```

35.4 Miscellaneous

\@@_iv_str_to_num:Nn This macro takes a four character string and converts it to the numerical representation required for XƎTEX OpenType script/language/feature purposes. The output is stored in #1.

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

```
\cs_set:Nn \@@_iv_str_to_num:Nn
```

```
 {
  \@@_iv_str_to_num:w #1 \q_nil #2 \c_empty_tl \c_empty_tl \q_nil
 }
\cs_set:Npn \@@_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 \@@_iv_str_to_num:Nn {No}
```

36 OpenType definitions code

fine_opentype_feature_group:n

```
\cs_new:Nn \@@_define_opentype_feature_group:n
  {
    \keys_define:nn {fontspec-opentype} { #1 .multichoice: }
  }
```

define_opentype_feature:nnnnn

#1 : Feature key
#2 : Feature option val
#3 : Check feature — leave empty for no check
#4 : Exact tag string to activate — leave empty for disable only
#5 : Tags to remove (clist)

```
\cs_new:Nn \@@_feat_prop_add:nn
  {
    \tl_if_empty:nF {#1}
     {
      \prop_if_in:NnF \g_@@_OT_features_prop {#1}
        {
          \prop_gput:Nnn \g_@@_OT_features_prop {#1} {#2}
        }
     }
  }
\cs_new:Nn \@@_define_opentype_feature:nnnnn
 {
  \@@_feat_prop_add:nn {#3} {#1\,=\,#2}
  \keys_define:nn {fontspec-opentype}
   {
    #1/#2 .code:n = { \@@_make_OT_feature:nnn {#3} {#4} {#5} }
   }
 }
```

ine_opentype_onoffreset:nnnnn

#1 : Feature key
#2 : Feature option val
#3 : Check feature

#4 : Tag prefix to activate: +#4 = on, -#4 = off.
#5 : Tags to remove in the on case (clist)

```
\cs_new:Nn \@@_feat_off:n {#1Off}
\cs_new:Nn \@@_feat_reset:n {#1Reset}
```

```
\cs_new:Nn \@@_define_opentype_onoffreset:nnnnn
 {
  \exp_args:Nnx \@@_define_opentype_feature:nnnnn {#1} {#2} {#3} {+#4} {#5}
  \exp_args:Nnx \@@_define_opentype_feature:nnnnn {#1} { \@@_feat_off:n   {#2} } {#3} {-#4} {}
  \exp_args:Nnx \@@_define_opentype_feature:nnnnn {#1} { \@@_feat_reset:n {#2} } {} {} {+#4,-#4}
 }
```

define_opentype_onreset:nnnnn

#1 : Feature key
#2 : Feature option val
#3 : Check feature
#4 : Exact tag string to activate
#5 : Tags to remove (clist)

```
\cs_new:Nn \@@_define_opentype_onreset:nnnnn
 {
  \exp_args:Nnx \@@_define_opentype_feature:nnnnn {#1} {#2} {#3} {#4} {#5}
  \exp_args:Nnx \@@_define_opentype_feature:nnnnn {#1} { \@@_feat_reset:n {#2} } {} {} {#4}
 }
```

36.1 Adding features when loading fonts

When remove clashing features,

1. remove the feature being added (to avoid duplicates);
2. remove the inverse of the feature (to avoid cancellation);
3. finally remove all clashing features.

```
\cs_new:Nn \@@_make_OT_feature:nnn
  {
⟨debug⟩  \typeout{:: @@_make_OT_feature:nnn \exp_not:n { {#1}{#2}{#3} } }

    \bool_set_true:N \l_@@_proceed_bool
    \bool_set_true:N  \l_@@_check_feat_bool

    \tl_if_empty:nT {#1} { \bool_set_false:N \l_@@_check_feat_bool }
    \bool_if:NT \l_@@_check_feat_bool
      {
        \@@_check_ot_feat:nF {#1}
          {
            \@@_warning:nx {icu-feature-not-exist-in-font} {#1}
            \bool_set_false:N \l_@@_proceed_bool
          }
      }

    \bool_if:NT \l_@@_proceed_bool
      {
        \exp_args:Nx \@@_remove_clashing_featstr:n
          { #2 , \@@_swap_plus_minus:n {#2} , #3 }

        \@@_update_featstr:n {#2}
      }
  }
\cs_generate_variant:Nn \@@_make_OT_feature:nnn {xxx}
```

```
\cs_new:Nn \@@_swap_plus_minus:n { \@@_swap_plus_minus_aux:NNNNN #1 }
\cs_new:Nn \@@_swap_plus_minus_aux:NNNNN
  { \str_case:nn {#1} { {+} {-#2#3#4#5} {-} {+#2#3#4#5} } }
```

\@@_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_@@_script_int is used to store the number corresponding to the script tag string.

```
\prg_new_conditional:Nnn \@@_check_script:n {TF}
  {
    \bool_if:NTF \l_@@_never_check_bool
      { \prg_return_true: }
⟨*xetexx⟩
 {
  \@@_iv_str_to_num:Nn \l_@@_strnum_int {#1}
  \int_set:Nn \l_tmpb_int { \XeTeXOTcountscripts \l_fontspec_font }
  \int_zero:N \l_tmpa_int
  \bool_set_false:N \l__fontspec_check_bool
  \bool_until_do:nn { \int_compare_p:nNn \l_tmpa_int = \l_tmpb_int }
   {
    \ifnum \XeTeXOTscripttag\l_fontspec_font \l_tmpa_int = \l_@@_strnum_int
      \bool_set_true:N \l__fontspec_check_bool
      \int_set:Nn \l_tmpa_int {\l_tmpb_int}
    \else
      \int_incr:N \l_tmpa_int
    \fi
   }
  \bool_if:NTF \l__fontspec_check_bool \prg_return_true: \prg_return_false:
 }
⟨/xetexx⟩
⟨*luatex⟩
 {
  \directlua{fontspec.check_ot_script("l_fontspec_font", "#1")}
  \bool_if:NTF \l__fontspec_check_bool \prg_return_true: \prg_return_false:
 }
⟨/luatex⟩
 }
```

\@@_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_@@_language_int is used to store the number corresponding to the language tag string. The script used is whatever's held in \l_@@_script_int. By default, that's the number corresponding to 'latn'.

```
\prg_new_conditional:Nnn \@@_check_lang:n {TF}
  {
    \bool_if:NTF \l_@@_never_check_bool
      { \prg_return_true: }
⟨*xetexx⟩
 {
  \@@_iv_str_to_num:Nn \l_@@_strnum_int {#1}
  \int_set:Nn \l_tmpb_int
   { \XeTeXOTcountlanguages \l_fontspec_font \l_@@_script_int }
  \int_zero:N \l_tmpa_int
  \bool_set_false:N \l__fontspec_check_bool
  \bool_until_do:nn { \int_compare_p:nNn \l_tmpa_int = \l_tmpb_int }
   {
    \ifnum\XeTeXOTlanguagetag\l_fontspec_font\l_@@_script_int \l_tmpa_int =\l_@@_strnum_int
      \bool_set_true:N \l__fontspec_check_bool
      \int_set:Nn \l_tmpa_int {\l_tmpb_int}
    \else
      \int_incr:N \l_tmpa_int
    \fi
   }
  \bool_if:NTF \l__fontspec_check_bool \prg_return_true: \prg_return_false:
 }
⟨/xetexx⟩
⟨*luatex⟩
 {
  \directlua
   {
    fontspec.check_ot_lang( "l_fontspec_font", "#1", "\l_fontspec_script_tl" )
   }
  \bool_if:NTF \l__fontspec_check_bool \prg_return_true: \prg_return_false:
 }
⟨/luatex⟩
  }
```

\@@_check_ot_feat:nTF This macro takes an OpenType feature tag and checks if it exists in the current font/script/language. \l_@@_strnum_int is used to store the number corresponding to the feature tag string. The script used is whatever's held in \l_@@_script_int. By default, that's the number corresponding to 'latn'. The language used is \l_@@_language_int, by default 0, the 'default language'.

```
\prg_new_conditional:Nnn \@@_check_ot_feat:n {TF,F}
  {
    \bool_if:NTF \l_@@_never_check_bool
      { \prg_return_true: }
⟨*xetexx⟩
 {
⟨debug⟩\typeout{::~ fontspec_check_ot_feat:n~ {#1}}
  \int_set:Nn \l_tmpb_int
   {
    \XeTeXOTcountfeatures \l_fontspec_font
                          \l_@@_script_int
                          \l_@@_language_int
  }
 \@@_iv_str_to_num:Nn \l_@@_strnum_int {#1}
 \int_zero:N \l_tmpa_int
 \bool_set_false:N \l_@@_check_bool
 \bool_until_do:nn { \int_compare_p:nNn \l_tmpa_int = \l_tmpb_int }
  {
   \ifnum\XeTeXOTfeaturetag\l_fontspec_font\l_@@_script_int\l_@@_language_int
        \l_tmpa_int =\l_@@_strnum_int
     \bool_set_true:N \l_@@_check_bool
     \int_set:Nn \l_tmpa_int {\l_tmpb_int}
   \else
     \int_incr:N \l_tmpa_int
   \fi
  }
 \bool_if:NTF \l_@@_check_bool \prg_return_true: \prg_return_false:
}
⟨/xetexx⟩
⟨*luatex⟩
{
⟨debug⟩\typeout{::~ fontspec_check_ot_feat:n~ {#1}}
 \directlua
  {
   fontspec.check_ot_feat(
                          "l_fontspec_font", "#1",
                          "\l_fontspec_lang_tl", "\l_fontspec_script_tl"
                         )
  }
 \bool_if:NTF \l_@@_check_bool \prg_return_true: \prg_return_false:
}
⟨/luatex⟩
 }
```

36.2 OpenType feature information

```
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {aalt}{Access~All~Alternates}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {abvf}{Above-base~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {abvm}{Above-base~Mark~Positioning}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {abvs}{Above-base~Substitutions}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {afrc}{Alternative~Fractions}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {akhn}{Akhands}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {blwf}{Below-base~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {blwm}{Below-base~Mark~Positioning}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {blws}{Below-base~Substitutions}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {calt}{Contextual~Alternates}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {case}{Case-Sensitive~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {ccmp}{Glyph~Composition~/~Decomposition
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {cfar}{Conjunct~Form~After~Ro}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {cjct}{Conjunct~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {clig}{Contextual~Ligatures}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {cpct}{Centered~CJK~Punctuation}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {cpsp}{Capital~Spacing}
```

```
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {cswh}{Contextual~Swash}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {curs}{Cursive~Positioning}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {cvNN}{Character~Variant~$N$}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {c2pc}{Petite~Capitals~From~Capitals}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {c2sc}{Small~Capitals~From~Capitals}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {dist}{Distances}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {dlig}{Discretionary~Ligatures}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {dnom}{Denominators}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {dtls}{Dotless~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {expt}{Expert~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {falt}{Final~Glyph~on~Line~Alternates}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {fin2}{Terminal~Forms~\#2}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {fin3}{Terminal~Forms~\#3}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {fina}{Terminal~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {flac}{Flattened~accent~forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {frac}{Fractions}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {fwid}{Full~Widths}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {half}{Half~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {haln}{Halant~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {halt}{Alternate~Half~Widths}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {hist}{Historical~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {hkna}{Horizontal~Kana~Alternates}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {hlig}{Historical~Ligatures}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {hngl}{Hangul}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {hojo}{Hojo~Kanji~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {hwid}{Half~Widths}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {init}{Initial~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {isol}{Isolated~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {ital}{Italics}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {jalt}{Justification~Alternates}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {jp78}{JIS78~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {jp83}{JIS83~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {jp90}{JIS90~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {jp04}{JIS2004~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {kern}{Kerning}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {lfbd}{Left~Bounds}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {liga}{Standard~Ligatures}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {ljmo}{Leading~Jamo~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {lnum}{Lining~Figures}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {locl}{Localized~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {ltra}{Left-to-right~alternates}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {ltrm}{Left-to-right~mirrored~forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {mark}{Mark~Positioning}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {med2}{Medial~Forms~\#2}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {medi}{Medial~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {mgrk}{Mathematical~Greek}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {mkmk}{Mark~to~Mark~Positioning}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {mset}{Mark~Positioning~via~Substitution
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {nalt}{Alternate~Annotation~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {nlck}{NLC~Kanji~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {nukt}{Nukta~Forms}
```

```
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {numr}{Numerators}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {onum}{Oldstyle~Figures}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {opbd}{Optical~Bounds}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {ordn}{Ordinals}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {ornm}{Ornaments}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {palt}{Proportional~Alternate~Widths}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {pcap}{Petite~Capitals}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {pkna}{Proportional~Kana}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {pnum}{Proportional~Figures}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {pref}{Pre-Base~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {pres}{Pre-base~Substitutions}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {pstf}{Post-base~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {psts}{Post-base~Substitutions}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {pwid}{Proportional~Widths}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {qwid}{Quarter~Widths}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {rand}{Randomize}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {rclt}{Required~Contextual~Alternates}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {rkrf}{Rakar~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {rlig}{Required~Ligatures}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {rphf}{Reph~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {rtbd}{Right~Bounds}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {rtla}{Right-to-left~alternates}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {rtlm}{Right-to-left~mirrored~forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {ruby}{Ruby~Notation~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {rvrn}{Required~Variation~Alternates}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {salt}{Stylistic~Alternates}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {sinf}{Scientific~Inferiors}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {size}{Optical~size}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {smcp}{Small~Capitals}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {smpl}{Simplified~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {ssNN}{Stylistic~Set~$N$}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {ssty}{Math~script~style~alternates}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {stch}{Stretching~Glyph~Decomposition}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {subs}{Subscript}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {sups}{Superscript}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {swsh}{Swash}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {titl}{Titling}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {tjmo}{Trailing~Jamo~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {tnam}{Traditional~Name~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {tnum}{Tabular~Figures}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {trad}{Traditional~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {twid}{Third~Widths}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {unic}{Unicase}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {valt}{Alternate~Vertical~Metrics}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {vatu}{Vattu~Variants}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {vert}{Vertical~Writing}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {vhal}{Alternate~Vertical~Half~Metrics}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {vjmo}{Vowel~Jamo~Forms}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {vkna}{Vertical~Kana~Alternates}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {vkrn}{Vertical~Kerning}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {vpal}{Proportional~Alternate~Vertical~Me
```

```
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {vrt2}{Vertical~Alternates~and~Rotation}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {vrtr}{Vertical~Alternates~for~Rotation}
\prop_gput:Nnn \g_@@_all_opentype_feature_names_prop {zero}{Slashed~Zero}
```

37 Graphite/AAT code

@@_define_aat_feature_group:n

```
\cs_new:Nn \@@_define_aat_feature_group:n
  { \keys_define:nn {fontspec-aat} { #1 .multichoice: } }
```

\@@_define_aat_feature:nnnn

```
\cs_new:Nn \@@_define_aat_feature:nnnn
 {
  \keys_define:nn {fontspec-aat}
   {
    #1/#2 .code:n = { \@@_make_AAT_feature:nn {#3}{#4} }
   }
 }
```

\@@_make_AAT_feature:nn

```
\cs_new:Nn \@@_make_AAT_feature:nn
 {
  \tl_if_empty:nTF {#1}
   { \@@_warning:n {aat-feature-not-exist} }
   {
     \@@_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} }
   }
 }
```

_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 108).

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 \@@_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:
    }
  }
}
```

38 Font loading (keyval) definitions

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

```
\clist_set:Nn \g_@@_all_keyval_modules_clist
  {
    fontspec, fontspec-opentype, fontspec-aat,
    fontspec-preparse, fontspec-preparse-external, fontspec-preparse-nested,
    fontspec-renderer
  }
```

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

For catching features that cannot be used in \addfontfeatures:

```
\cs_new:Nn \@@_aff_error:n
  {
    \@@_keys_define_code:nnn {fontspec-addfeatures} {#1}
      { \@@_error:nx {not-in-addfontfeatures} {#1} }
  }
```

38.0.1 Pre-parsing naming information

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

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

```
\@@_keys_define_code:nnn {fontspec-preparse-external} {Path}
 {
  \bool_set_true:N \l_@@_nobf_bool
  \bool_set_true:N \l_@@_noit_bool
  \bool_set_true:N \l_@@_external_bool
  \tl_set:Nn \l_@@_font_path_tl {#1}
  \@@_font_is_file:
⟨*xetexx⟩
  \keys_set:nn {fontspec-renderer} {Renderer=OpenType}
⟨/xetexx⟩
 }
\aliasfontfeature{Path}{ExternalLocation}
\@@_keys_define_code:nnn {fontspec} {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} {Path}
   }
 }
\tl_clear:N \l_@@_extension_tl
\@@_keys_define_code:nnn {fontspec} {Extension} {}
```

38.0.2 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 }
      \tl_gset:Nx \g_@@_single_feat_tl { \l_fontspec_renderer_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 }
      \tl_gset:Nx \g_@@_single_feat_tl { mode=\l_fontspec_mode_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}
 }
```

TTC font index

```
\@@_keys_define_code:nnn {fontspec-preparse} {FontIndex}
 {
  \str_if_eq_x:nnF { \str_lower_case:f {\l_@@_extension_tl} } {.ttc}
    { \@@_warning:n {font-index-needs-ttc} }
⟨xetexx⟩  \tl_set:Nn \l_@@_ttc_index_tl {:#1}
⟨luatex⟩  \tl_set:Nn \l_@@_ttc_index_tl {(#1)}
 }
\@@_keys_define_code:nnn {fontspec} {FontIndex}
 {
```

```
⟨xetexx⟩   \tl_set:Nn \l_@@_ttc_index_tl {:#1}
⟨luatex⟩   \tl_set:Nn \l_@@_ttc_index_tl {(#1)}
 }
```

38.0.3 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.

```
\@@_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_@@_fontname_up_tl {#1}
 }
\@@_keys_define_code:nnn {fontspec-preparse-external} {FontName}
 {
  \fontspec_complete_fontname:Nn \l_@@_fontname_up_tl {#1}
 }
```

Bold:

```
\@@_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_@@_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

   }
 }
```

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_@@_fontname_it_tl {#1}
   }
 }
```

Simpler for bold+italic & slanted:

```
\@@_keys_define_code:nnn {fontspec-preparse-external} {BoldItalicFont}
 {
  \fontspec_complete_fontname:Nn \l_@@_fontname_bfit_tl {#1}
 }
\@@_keys_define_code:nnn {fontspec-preparse-external} {SlantedFont}
 {
  \fontspec_complete_fontname:Nn \l_@@_fontname_sl_tl {#1}
 }
\@@_keys_define_code:nnn {fontspec-preparse-external} {BoldSlantedFont}
 {
  \fontspec_complete_fontname:Nn \l_@@_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_@@_fontname_sc_tl {#1}
   }
 }
```

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}
 }
```

38.0.4 Font-independent features

These features can be applied to any font.

NFSS encoding For the very brave.

```
\@@_keys_define_code:nnn {fontspec-preparse} {NFSSEncoding}
 {
  \tl_gset:Nx \l_@@_nfss_enc_tl { #1 }
 }
```

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.

```
\@@_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:nnnn
  {\use_i:nnn #1}{\use_ii:nnn #1}{\l_@@_this_font_tl}{\l_@@_this_feat_tl}
}
```

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:nnF {#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 XƎTEX).

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:nnF #2
       {
         {5} {x} % x-height
         {8} {X} % cap-height
       } {?} % "else" clause; never reached.
     }
   }
 }
```

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

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

_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:nnF {#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 }
   }
 }
\@@_aff_error:n {PunctuationSpace}
```

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}
 {
  \bool_if:NT \l_@@_addfontfeatures_bool
   { \@@_error:nx {not-in-addfontfeatures} {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} }
    \@@_primitive_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
          \int_set:Nn \luatex_prehyphenchar:D { \l_fontspec_hyphenchar_tl }
        }
⟨/luatex⟩
     }
     { \@@_error:nx {no-glyph}{#1} }
   }
 }
\@@_aff_error:n {HyphenChar}
```

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}
```

```
\@@_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_hex: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-opentype} { Ligatures=TeX }
   }
   { \@@_warning:n {no-mapping} }
 }
⟨/luatex⟩
```

38.0.5 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⟩
```

38.0.6 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_@@_fake_slant_tl {#1}
      \clist_put_right:Nn \l_@@_fontfeat_it_clist {FakeSlant=#1}
      \tl_set_eq:NN \l_@@_fontname_it_tl \l_fontspec_fontname_tl
      \bool_set_false:N \l_@@_noit_bool

      \tl_if_empty:NF \l_@@_fake_embolden_tl
       {
        \clist_put_right:Nx \l_@@_fontfeat_bfit_clist
         {FakeBold=\l_@@_fake_embolden_tl}
        \clist_put_right:Nx \l_@@_fontfeat_bfit_clist {FakeSlant=#1}
        \tl_set_eq:NN \l_@@_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_@@_fake_embolden_tl {#1}
      \clist_put_right:Nn \l_@@_fontfeat_bf_clist {FakeBold=#1}
      \tl_set_eq:NN \l_@@_fontname_bf_tl \l_fontspec_fontname_tl
      \bool_set_false:N \l_@@_nobf_bool

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

38.0.7 Raw feature string

This allows savvy XƎTEX-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}
 }
```

38.1 OpenType feature definitions

```
\@@_feat_prop_add:nn {salt} { Alternate\,=\,$N$ }
\@@_feat_prop_add:nn {nalt} { Annotation\,=\,$N$ }
\@@_feat_prop_add:nn {ornm} { Ornament\,=\,$N$ }
\@@_feat_prop_add:nn {cvNN} { CharacterVariant\,=\,$N$:$M$ }
\@@_feat_prop_add:nn {ssNN} { StylisticSet\,=\,$N$ }
```

38.2 Regular key=val / tag definitions

38.2.1 Ligatures

```
\@@_define_opentype_feature_group:n {Ligatures}
\@@_define_opentype_feature:nnnnn {Ligatures} {ResetAll} {} {}
  {
    +dlig,-dlig,+rlig,-rlig,+liga,-liga,+dlig,-dlig,+clig,-clig,+hlig,-hlig,
⟨xetexx⟩  mapping = tex-text
⟨luatex⟩  +tlig,-tlig
  }
```

```
\@@_define_opentype_onoffreset:nnnnn {Ligatures} {Required}      {rlig} {rlig} {}
\@@_define_opentype_onoffreset:nnnnn {Ligatures} {Common}        {liga} {liga} {}
\@@_define_opentype_onoffreset:nnnnn {Ligatures} {Rare}          {dlig} {dlig} {}
\@@_define_opentype_onoffreset:nnnnn {Ligatures} {Discretionary} {dlig} {dlig} {}
\@@_define_opentype_onoffreset:nnnnn {Ligatures} {Contextual}    {clig} {clig} {}
\@@_define_opentype_onoffreset:nnnnn {Ligatures} {Historic}      {hlig} {hlig} {}
```

Emulate CM extra ligatures.

```
⟨xetexx⟩\@@_define_opentype_onreset:nnnnn {Ligatures} {TeX} {} { mapping = tex-text } {}
⟨luatex⟩\@@_define_opentype_onreset:nnnnn {Ligatures} {TeX} {} { +tlig } {}
```

38.2.2 Letters

```
\@@_define_opentype_feature_group:n {Letters}
\@@_define_opentype_feature:nnnnn   {Letters} {ResetAll} {} {}
  {
    +case,+smcp,+pcap,+c2sc,+c2pc,+unic,+rand,
    -case,-smcp,-pcap,-c2sc,-c2pc,-unic,-rand
  }
```

```
\@@_define_opentype_onoffreset:nnnnn {Letters} {Uppercase} {case} {case} {+smcp,+pcap,+c2sc,+c
\@@_define_opentype_onoffreset:nnnnn {Letters} {SmallCaps} {smcp} {smcp} {+pcap,+unic,+rand}
\@@_define_opentype_onoffreset:nnnnn {Letters} {PetiteCaps} {pcap} {pcap} {+smcp,+unic,+rand}
```

```
\@@_define_opentype_onoffreset:nnnnn {Letters} {UppercaseSmallCaps} {c2sc} {c2sc} {+c2pc,+unic
\@@_define_opentype_onoffreset:nnnnn {Letters} {UppercasePetiteCaps} {c2pc} {c2pc} {+c2sc,+uni
\@@_define_opentype_onoffreset:nnnnn {Letters} {Unicase} {unic} {unic} {+rand}
\@@_define_opentype_onoffreset:nnnnn {Letters} {Random} {rand} {rand} {+unic}
```

38.2.3 Numbers

```
\@@_define_opentype_feature_group:n {Numbers}
\@@_define_opentype_feature:nnnnn   {Numbers} {ResetAll} {} {}
  {
    +tnum,-tnum,
    +pnum,-pnum,
    +onum,-onum,
    +lnum,-lnum,
    +zero,-zero,
    +anum,-anum,
  }
```

```
\@@_define_opentype_onoffreset:nnnnn {Numbers} {Monospaced}   {tnum} {tnum} {+pnum,-pnum}
\@@_define_opentype_onoffreset:nnnnn {Numbers} {Proportional} {pnum} {pnum} {+tnum,-tnum}
\@@_define_opentype_onoffreset:nnnnn {Numbers} {Lowercase}    {onum} {onum} {+lnum,-lnum}
\@@_define_opentype_onoffreset:nnnnn {Numbers} {Uppercase}    {lnum} {lnum} {+onum,-onum}
\@@_define_opentype_onoffreset:nnnnn {Numbers} {SlashedZero}  {zero} {zero} {}
```

```
\aliasfontfeatureoption {Numbers} {Lowercase} {OldStyle}
\aliasfontfeatureoption {Numbers} {Uppercase} {Lining}
```

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⟩  \@@_define_opentype_onoffreset:nnnnn {Numbers} {Arabic} {anum} {anum} {}
```

38.2.4 Vertical position

```
\@@_define_opentype_feature_group:n  {VerticalPosition}
\@@_define_opentype_feature:nnnnn    {VerticalPosition} {ResetAll} {} {}
  {
    +sups,-sups,
    +subs,-subs,
    +ordn,-ordn,
    +numr,-numr,
    +dnom,-dnom,
    +sinf,-sinf,
  }
```

```
\@@_define_opentype_onoffreset:nnnnn {VerticalPosition} {Superior}          {sups} {sups} {+s
\@@_define_opentype_onoffreset:nnnnn {VerticalPosition} {Inferior}          {subs} {subs} {+s
\@@_define_opentype_onoffreset:nnnnn {VerticalPosition} {Ordinal}           {ordn} {ordn} {+s
\@@_define_opentype_onoffreset:nnnnn {VerticalPosition} {Numerator}         {numr} {numr} {+s
\@@_define_opentype_onoffreset:nnnnn {VerticalPosition} {Denominator}       {dnom} {dnom} {+s
\@@_define_opentype_onoffreset:nnnnn {VerticalPosition} {ScientificInferior} {sinf} {sinf} {+s
```

38.2.5 Contextuals

```
\@@_define_opentype_feature_group:n  {Contextuals}
\@@_define_opentype_feature:nnnnn    {Contextuals} {ResetAll} {} {}
  {
    +cswh,-cswh,
    +calt,-calt,
    +init,-init,
    +fina,-fina,
    +falt,-falt,
    +medi,-medi,
  }
\@@_define_opentype_onoffreset:nnnnn {Contextuals} {Swash}       {cswh} {cswh} {}
\@@_define_opentype_onoffreset:nnnnn {Contextuals} {Alternate}   {calt} {calt} {}
\@@_define_opentype_onoffreset:nnnnn {Contextuals} {WordInitial} {init} {init} {}
\@@_define_opentype_onoffreset:nnnnn {Contextuals} {WordFinal}   {fina} {fina} {}
\@@_define_opentype_onoffreset:nnnnn {Contextuals} {LineFinal}   {falt} {falt} {}
\@@_define_opentype_onoffreset:nnnnn {Contextuals} {Inner}       {medi} {medi} {}
```

38.2.6 Diacritics

```
\@@_define_opentype_feature_group:n  {Diacritics}
\@@_define_opentype_feature:nnnnn    {Diacritics} {ResetAll} {} {}
  {
    +mark,-mark,
    +mkmk,-mkmk,
    +abvm,-abvm,
    +blwm,-blwm,
  }
\@@_define_opentype_onoffreset:nnnnn {Diacritics} {MarkToBase} {mark} {mark} {}
\@@_define_opentype_onoffreset:nnnnn {Diacritics} {MarkToMark} {mkmk} {mkmk} {}
\@@_define_opentype_onoffreset:nnnnn {Diacritics} {AboveBase}  {abvm} {abvm} {}
\@@_define_opentype_onoffreset:nnnnn {Diacritics} {BelowBase}  {blwm} {blwm} {}
```

38.2.7 Kerning

```
\@@_define_opentype_feature_group:n  {Kerning}
\@@_define_opentype_feature:nnnnn    {Kerning} {ResetAll} {} {}
  {
    +cpsp,-cpsp,
    +kern,-kern,
  }
\@@_define_opentype_onoffreset:nnnnn {Kerning} {Uppercase} {cpsp} {cpsp} {}
\@@_define_opentype_feature:nnnnn    {Kerning} {On}        {kern} {+kern} {-kern}
\@@_define_opentype_feature:nnnnn    {Kerning} {Off}       {kern} {-kern} {+kern}
\@@_define_opentype_feature:nnnnn    {Kerning} {Reset}     {} {} {+kern,-kern}
```

38.2.8 Fractions

```
\@@_define_opentype_feature_group:n  {Fractions}
\@@_define_opentype_feature:nnnnn    {Fractions} {ResetAll} {} {}
  {
    +frac,-frac,
    +afrc,-afrc,
  }
```

```
\@@_define_opentype_feature:nnnnn    {Fractions} {On}    {frac} {+frac} {}
\@@_define_opentype_feature:nnnnn    {Fractions} {Off}   {frac} {-frac} {}
\@@_define_opentype_feature:nnnnn    {Fractions} {Reset} {} {} {+frac,-frac}
\@@_define_opentype_onoffreset:nnnnn {Fractions} {Alternate} {afrc} {afrc} {-frac}
```

38.2.9 Style

```
\@@_define_opentype_feature_group:n  {Style}
\@@_define_opentype_feature:nnnnn    {Style} {ResetAll} {} {}
  {
    +salt,-salt,
    +ital,-ital,
    +ruby,-ruby,
    +swsh,-swsh,
    +hist,-hist,
    +titl,-titl,
    +hkna,-hkna,
    +vkna,-vkna,
    +ssty=0,-ssty=0,
    +ssty=1,-ssty=1,
  }
```

```
\@@_define_opentype_onoffreset:nnnnn {Style} {Alternate}        {salt} {salt} {}
\@@_define_opentype_onoffreset:nnnnn {Style} {Italic}           {ital} {ital} {}
\@@_define_opentype_onoffreset:nnnnn {Style} {Ruby}             {ruby} {ruby} {}
\@@_define_opentype_onoffreset:nnnnn {Style} {Swash}            {swsh} {swsh} {}
\@@_define_opentype_onoffreset:nnnnn {Style} {Cursive}          {swsh} {curs} {}
\@@_define_opentype_onoffreset:nnnnn {Style} {Historic}         {hist} {hist} {}
\@@_define_opentype_onoffreset:nnnnn {Style} {TitlingCaps}      {titl} {titl} {}
\@@_define_opentype_onoffreset:nnnnn {Style} {HorizontalKana}   {hkna} {hkna} {+vkna,+pkna}
\@@_define_opentype_onoffreset:nnnnn {Style} {VerticalKana}     {vkna} {vkna} {+hkna,+pkna}
\@@_define_opentype_onoffreset:nnnnn {Style} {ProportionalKana} {pkna} {pkna} {+vkna,+hkna}
\@@_define_opentype_feature:nnnnn    {Style} {MathScript}       {ssty} {+ssty=0} {+ssty=1}
\@@_define_opentype_feature:nnnnn    {Style} {MathScriptScript} {ssty} {+ssty=1} {+ssty=0}
```

38.2.10 CJK shape

```
\@@_define_opentype_feature_group:n  {CJKShape}
\@@_define_opentype_feature:nnnnn    {CJKShape} {ResetAll} {} {}
  {
    +trad,-trad,
    +smpl,-smpl,
    +jp78,-jp78,
    +jp83,-jp83,
    +jp90,-jp90,
    +jp04,-jp04,
    +expt,-expt,
    +nlck,-nlck,
  }
```

```
\@@_define_opentype_onoffreset:nnnnn {CJKShape} {Traditional} {trad} {trad} {+smpl,+jp78,+jp8
```

```
\@@_define_opentype_onoffreset:nnnnn {CJKShape} {Simplified}  {smpl} {smpl} {+trad,+jp78,+jp8
\@@_define_opentype_onoffreset:nnnnn {CJKShape} {JIS1978}     {jp78} {jp78} {+trad,+smpl,+jp8
\@@_define_opentype_onoffreset:nnnnn {CJKShape} {JIS1983}     {jp83} {jp83} {+trad,+smpl,+jp7
\@@_define_opentype_onoffreset:nnnnn {CJKShape} {JIS1990}     {jp90} {jp90} {+trad,+smpl,+jp7
\@@_define_opentype_onoffreset:nnnnn {CJKShape} {JIS2004}     {jp04} {jp04} {+trad,+smpl,+jp7
\@@_define_opentype_onoffreset:nnnnn {CJKShape} {Expert}      {expt} {expt} {+trad,+smpl,+jp7
\@@_define_opentype_onoffreset:nnnnn {CJKShape} {NLC}         {nlck} {nlck} {+trad,+smpl,+jp7
```

38.2.11 Character width

```
\@@_define_opentype_feature_group:n  {CharacterWidth}
\@@_define_opentype_feature:nnnnn    {CharacterWidth} {ResetAll} {} {}
  {
    +pwid,-pwid,
    +fwid,-fwid,
    +hwid,-hwid,
    +twid,-twid,
    +qwid,-qwid,
    +palt,-palt,
    +halt,-halt,
  }
```

```
\@@_define_opentype_onoffreset:nnnnn {CharacterWidth} {Proportional}          {pwid} {pwid} {-
\@@_define_opentype_onoffreset:nnnnn {CharacterWidth} {Full}                  {fwid} {fwid} {-
\@@_define_opentype_onoffreset:nnnnn {CharacterWidth} {Half}                  {hwid} {hwid} {-
\@@_define_opentype_onoffreset:nnnnn {CharacterWidth} {Third}                 {twid} {twid} {-
\@@_define_opentype_onoffreset:nnnnn {CharacterWidth} {Quarter}               {qwid} {qwid} {-
\@@_define_opentype_onoffreset:nnnnn {CharacterWidth} {AlternateProportional} {palt} {palt} {-
\@@_define_opentype_onoffreset:nnnnn {CharacterWidth} {AlternateHalf}         {halt} {halt} {-
```

38.2.12 Vertical

According to spec vkrn must also activate vpal if available but for simplicity we don't do that here (yet?).

```
\@@_define_opentype_feature_group:n {Vertical}
\@@_define_opentype_onoffreset:nnnnn {Vertical} {RotatedGlyphs}         {vrt2} {vrt2} {+vrtr,-
\@@_define_opentype_onoffreset:nnnnn {Vertical} {AlternatesForRotation} {vrtr} {vrtr} {+vrt2}
\@@_define_opentype_onoffreset:nnnnn {Vertical} {Alternates}            {vert} {vert} {+vrt2}
\@@_define_opentype_onoffreset:nnnnn {Vertical} {KanaAlternates}        {vkna} {vkna} {+hkna}
\@@_define_opentype_onoffreset:nnnnn {Vertical} {Kerning}               {vkrn} {vkrn} {}
\@@_define_opentype_onoffreset:nnnnn {Vertical} {AlternateMetrics}      {valt} {valt} {+vhal,-
\@@_define_opentype_onoffreset:nnnnn {Vertical} {HalfMetrics}           {vhal} {vhal} {+valt,-
\@@_define_opentype_onoffreset:nnnnn {Vertical} {ProportionalMetrics}   {vpal} {vpal} {+valt,-
```

38.3 OpenType features that need numbering

38.3.1 Alternate

```
\@@_define_opentype_feature_group:n  {Alternate}
\keys_define:nn {fontspec-opentype}
 {
  Alternate .default:n = {0} ,
⟨luatex⟩  Alternate / Random  .code:n =
⟨luatex⟩   { \@@_make_OT_feature:nnn {salt}{ +salt = random }{} } ,
  Alternate / unknown .code:n =
   {
    \clist_map_inline:nn {#1}
      { \@@_make_OT_feature:nnn {salt}{ +salt = ##1 }{} }
   }
 }
\aliasfontfeature{Alternate}{StylisticAlternates}
```

38.3.2 Variant / StylisticSet

```
\@@_define_opentype_feature_group:n  {Variant}
\keys_define:nn {fontspec-opentype}
 {
  Variant .default:n = {0} ,
  Variant / unknown .code:n =
   {
    \clist_map_inline:nn {#1}
      {
        \@@_make_OT_feature:xxx { ss \two@digits {##1} } { +ss \two@digits {##1} } {}
      }
   }
 }
\aliasfontfeature{Variant}{StylisticSet}
```

38.3.3 CharacterVariant

```
\@@_define_opentype_feature_group: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
   {
     \@@_make_OT_feature:xxx
       { cv \exp_not:N \two@digits {##1} } { +cv \exp_not:N \two@digits {##1} = ##2 } {}
   }
   \keys_define:nn {fontspec-opentype}
   {
    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.

38.3.4 Annotation

```
\@@_define_opentype_feature_group:n {Annotation}
\keys_define:nn {fontspec-opentype}
 {
  Annotation .default:n = {0} ,
  Annotation / unknown .code:n =
   {
    \@@_make_OT_feature:nnn {nalt} {+nalt=#1} {}
   }
 }
```

38.3.5 Ornament

```
\@@_define_opentype_feature_group:n  {Ornament}
\keys_define:nn {fontspec-opentype}
 {
  Ornament .default:n = {0} ,
  Ornament / unknown .code:n =
   {
    \@@_make_OT_feature:nnn {ornm} { +ornm=#1 } {}
   }
 }
```

38.4 Script and Language

38.4.1 Script

```
\keys_define:nn { fontspec-opentype } { Script .choice: }
\cs_new:Nn \fontspec_new_script:nn
 {
  \keys_define:nn { fontspec-opentype } { Script / #1 .code:n =
    \bool_set_false:N \l_@@_script_exist_bool
    \clist_map_inline:nn {#2}
     {
      \@@_check_script:nTF {####1}
       {
        \tl_set:Nn \l_fontspec_script_tl {####1}
        \int_set:Nn \l_@@_script_int {\l_@@_strnum_int}
        \bool_set_true:N \l_@@_script_exist_bool
        \tl_gset:Nx \g_@@_single_feat_tl { script=####1 }
        \clist_map_break:
       }
       { }
     }
    \bool_if:NF \l_@@_script_exist_bool
     {
      \str_if_eq:nnTF {#1} {Latin}
       {
        \@@_warning:nx {script-not-exist} {#1}
       }
       {
        \@@_check_script:nTF {latn}
         {
          \@@_warning:nx {script-not-exist-latn} {#1}
          \tl_set:Nn \l_fontspec_script_tl {latn}
          \int_set:Nn \l_@@_script_int {\l_@@_strnum_int}
         }
         {
          \@@_warning:nx {script-not-exist} {#1}
         }
       }
     }
   }
 }
```

38.4.2 Language

```
\keys_define:nn { fontspec-opentype } { Language .choice: }
\cs_new:Nn \fontspec_new_lang:nn
 {
  \keys_define:nn { fontspec-opentype } { Language / #1 .code:n =
  \@@_check_lang:nTF {#2}
     {
      \tl_set:Nn \l_fontspec_lang_tl {#2}
      \int_set:Nn \l_@@_language_int {\l_@@_strnum_int}
      \tl_gset:Nx \g_@@_single_feat_tl { language=#2 }
     }
     {
      \@@_warning:nx {language-not-exist} {#1}
      \keys_set:nn { fontspec-opentype } { Language = Default }
     }
  }
 }
```

Default

```
\@@_keys_define_code:nnn {fontspec-opentype}{ Language / Default }
 {
  \tl_set:Nn \l_fontspec_lang_tl {DFLT}
  \int_zero:N \l_@@_language_int
  \tl_gset:Nn \g_@@_single_feat_tl { language=DFLT }
 }
```

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-opentype}
 {
  Language / Turkish .code:n =
   {
    \@@_check_lang:nTF {TRK}
     {
      \int_set:Nn \l_@@_language_int {\l_@@_strnum_int}
      \tl_set:Nn \l_fontspec_lang_tl {TRK}
      \tl_gset:Nn \g_@@_single_feat_tl { language=TRK }
     }
     {
      \@@_check_lang:nTF {TUR}
       {
        \int_set:Nn \l_@@_language_int {\l_@@_strnum_int}
        \tl_set:Nn \l_fontspec_lang_tl {TUR}
        \tl_gset:Nn \g_@@_single_feat_tl { language=TUR }
        }
        {
         \@@_warning:nx {language-not-exist} {Turkish}
         \keys_set:nn {fontspec-opentype} {Language=Default}
        }
      }
    }
}
```

38.5 Backwards compatibility

Backwards compatibility:

```
\cs_new:Nn \@@_ot_compat:nn
  {
    \aliasfontfeatureoption {#1} {#2Off} {No#2}
  }
\@@_ot_compat:nn {Ligatures}   {Rare}
\@@_ot_compat:nn {Ligatures}   {Required}
\@@_ot_compat:nn {Ligatures}   {Common}
\@@_ot_compat:nn {Ligatures}   {Discretionary}
\@@_ot_compat:nn {Ligatures}   {Contextual}
\@@_ot_compat:nn {Ligatures}   {Historic}
\@@_ot_compat:nn {Numbers}     {SlashedZero}
\@@_ot_compat:nn {Contextuals} {Swash}
\@@_ot_compat:nn {Contextuals} {Alternate}
\@@_ot_compat:nn {Contextuals} {WordInitial}
\@@_ot_compat:nn {Contextuals} {WordFinal}
\@@_ot_compat:nn {Contextuals} {LineFinal}
\@@_ot_compat:nn {Contextuals} {Inner}
\@@_ot_compat:nn {Diacritics}  {MarkToBase}
\@@_ot_compat:nn {Diacritics}  {MarkToMark}
\@@_ot_compat:nn {Diacritics}  {AboveBase}
\@@_ot_compat:nn {Diacritics}  {BelowBase}
```

38.6 Font script definitions

```
\newfontscript{Arabic}{arab}
\newfontscript{Armenian}{armn}
\newfontscript{Balinese}{bali}
\newfontscript{Bengali}{bng2,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}{dev2,deva}
\newfontscript{Ethiopic}{ethi}
\newfontscript{Georgian}{geor}
\newfontscript{Glagolitic}{glag}
\newfontscript{Gothic}{goth}
\newfontscript{Greek}{grek}
\newfontscript{Gujarati}{gjr2,gujr}
\newfontscript{Gurmukhi}{gur2,guru}
\newfontscript{Hangul~Jamo}{jamo}
\newfontscript{Hangul}{hang}
\newfontscript{Hanunoo}{hano}
\newfontscript{Hebrew}{hebr}
\newfontscript{Hiragana~and~Katakana}{kana}
\newfontscript{Javanese}{java}
\newfontscript{Kannada}{knd2,knda}
\newfontscript{Kharosthi}{khar}
\newfontscript{Khmer}{khmr}
\newfontscript{Lao}{lao~}
\newfontscript{Latin}{latn}
\newfontscript{Limbu}{limb}
\newfontscript{Linear~B}{linb}
\newfontscript{Malayalam}{mlm2,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}{ory2,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}{tml2,taml}
\newfontscript{Telugu}{tel2,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}
```

38.7 Font language definitions

```
\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}
```

38.8 AAT feature definitions

These are only defined for XeTeX.

38.8.1 Ligatures

```
\@@_define_aat_feature_group:n {Ligatures}
\@@_define_aat_feature:nnnn      {Ligatures} {Required} {1} {0}
\@@_define_aat_feature:nnnn      {Ligatures} {NoRequired} {1} {1}
\@@_define_aat_feature:nnnn      {Ligatures} {Common} {1} {2}
\@@_define_aat_feature:nnnn      {Ligatures} {NoCommon} {1} {3}
\@@_define_aat_feature:nnnn      {Ligatures} {Rare} {1} {4}
\@@_define_aat_feature:nnnn      {Ligatures} {NoRare} {1} {5}
\@@_define_aat_feature:nnnn      {Ligatures} {Discretionary} {1} {4}
\@@_define_aat_feature:nnnn      {Ligatures} {NoDiscretionary} {1} {5}
\@@_define_aat_feature:nnnn      {Ligatures} {Logos} {1} {6}
\@@_define_aat_feature:nnnn      {Ligatures} {NoLogos} {1} {7}
```

```
\@@_define_aat_feature:nnnn      {Ligatures} {Rebus} {1} {8}
\@@_define_aat_feature:nnnn      {Ligatures} {NoRebus} {1} {9}
\@@_define_aat_feature:nnnn      {Ligatures} {Diphthong} {1} {10}
\@@_define_aat_feature:nnnn      {Ligatures} {NoDiphthong} {1} {11}
\@@_define_aat_feature:nnnn      {Ligatures} {Squared} {1} {12}
\@@_define_aat_feature:nnnn      {Ligatures} {NoSquared} {1} {13}
\@@_define_aat_feature:nnnn      {Ligatures} {AbbrevSquared} {1} {14}
\@@_define_aat_feature:nnnn      {Ligatures} {NoAbbrevSquared} {1} {15}
\@@_define_aat_feature:nnnn      {Ligatures} {Icelandic} {1} {32}
\@@_define_aat_feature:nnnn      {Ligatures} {NoIcelandic} {1} {33}
```

Emulate CM extra ligatures.

```
\keys_define:nn {fontspec-aat}
 {
  Ligatures / TeX .code:n =
   {
    \@@_update_featstr:n { mapping = tex-text }
   }
 }
```

38.8.2 Letters

```
\@@_define_aat_feature_group:n {Letters}
\@@_define_aat_feature:nnnn      {Letters} {Normal} {3} {0}
\@@_define_aat_feature:nnnn      {Letters} {Uppercase} {3} {1}
\@@_define_aat_feature:nnnn      {Letters} {Lowercase} {3} {2}
\@@_define_aat_feature:nnnn      {Letters} {SmallCaps} {3} {3}
\@@_define_aat_feature:nnnn      {Letters} {InitialCaps} {3} {4}
```

38.8.3 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_aat_feature_group:n {Numbers}
\@@_define_aat_feature:nnnn      {Numbers} {Monospaced} {6} {0}
\@@_define_aat_feature:nnnn      {Numbers} {Proportional} {6} {1}
\@@_define_aat_feature:nnnn      {Numbers} {Lowercase} {21} {0}
\@@_define_aat_feature:nnnn      {Numbers} {OldStyle} {21} {0}
\@@_define_aat_feature:nnnn      {Numbers} {Uppercase} {21} {1}
\@@_define_aat_feature:nnnn      {Numbers} {Lining} {21} {1}
\@@_define_aat_feature:nnnn      {Numbers} {SlashedZero} {14} {5}
\@@_define_aat_feature:nnnn      {Numbers} {NoSlashedZero} {14} {4}
```

38.8.4 Contextuals

```
\@@_define_aat_feature_group:n   {Contextuals}
\@@_define_aat_feature:nnnn      {Contextuals} {WordInitial} {8} {0}
\@@_define_aat_feature:nnnn      {Contextuals} {NoWordInitial} {8} {1}
\@@_define_aat_feature:nnnn      {Contextuals} {WordFinal} {8} {2}
\@@_define_aat_feature:nnnn      {Contextuals} {NoWordFinal} {8} {3}
\@@_define_aat_feature:nnnn      {Contextuals} {LineInitial} {8} {4}
\@@_define_aat_feature:nnnn      {Contextuals} {NoLineInitial} {8} {5}
```

```
\@@_define_aat_feature:nnnn      {Contextuals} {LineFinal} {8} {6}
\@@_define_aat_feature:nnnn      {Contextuals} {NoLineFinal} {8} {7}
\@@_define_aat_feature:nnnn      {Contextuals} {Inner} {8} {8}
\@@_define_aat_feature:nnnn      {Contextuals} {NoInner} {8} {9}
```

38.8.5 Diacritics

```
\@@_define_aat_feature_group:n {Diacritics}
\@@_define_aat_feature:nnnn      {Diacritics} {Show} {9} {0}
\@@_define_aat_feature:nnnn      {Diacritics} {Hide} {9} {1}
\@@_define_aat_feature:nnnn      {Diacritics} {Decompose} {9} {2}
```

38.8.6 Vertical position

```
\@@_define_aat_feature_group:n {VerticalPosition}
\@@_define_aat_feature:nnnn      {VerticalPosition} {Normal} {10} {0}
\@@_define_aat_feature:nnnn      {VerticalPosition} {Superior} {10} {1}
\@@_define_aat_feature:nnnn      {VerticalPosition} {Inferior} {10} {2}
\@@_define_aat_feature:nnnn      {VerticalPosition} {Ordinal} {10} {3}
```

38.8.7 Fractions

```
\@@_define_aat_feature_group:n {Fractions}
\@@_define_aat_feature:nnnn      {Fractions} {On} {11} {1}
\@@_define_aat_feature:nnnn      {Fractions} {Off} {11} {0}
\@@_define_aat_feature:nnnn      {Fractions} {Diagonal} {11} {2}
```

38.8.8 Alternate

```
\@@_define_aat_feature_group:n  { Alternate }
\keys_define:nn {fontspec-aat}
 {
  Alternate .default:n = {0} ,
  Alternate / unknown .code:n =
   {
    \clist_map_inline:nn {#1}
      {
        \@@_make_AAT_feature:nn {17}{##1}
      }
   }
 }
```

38.8.9 Variant / StylisticSet

```
\@@_define_aat_feature_group:n  {Variant}
\keys_define:nn {fontspec-aat}
 {
  Variant .default:n = {0} ,
  Variant / unknown .code:n =
   {
    \clist_map_inline:nn {#1}
      { \@@_make_AAT_feature:nn {18}{##1} }
   }
 }
\aliasfontfeature{Variant}{StylisticSet}
```

```
\@@_define_aat_feature_group:n  {Vertical}
```

```
\keys_define:nn {fontspec-aat}
 {
  Vertical .choice: ,
  Vertical / RotatedGlyphs .code:n =
    {
      \__fontspec_update_featstr:n {vertical}
    }
 }

```

38.8.10 Style

```
\@@_define_aat_feature_group:n {Style}
\@@_define_aat_feature:nnnn      {Style} {Italic} {32} {2}
\@@_define_aat_feature:nnnn      {Style} {Ruby} {28} {2}
\@@_define_aat_feature:nnnn      {Style} {Display} {19} {1}
\@@_define_aat_feature:nnnn      {Style} {Engraved} {19} {2}
\@@_define_aat_feature:nnnn      {Style} {TitlingCaps} {19} {4}
\@@_define_aat_feature:nnnn      {Style} {TallCaps} {19} {5}
```

38.8.11 CJK shape

```
\@@_define_aat_feature_group:n {CJKShape}
\@@_define_aat_feature:nnnn      {CJKShape} {Traditional} {20} {0}
\@@_define_aat_feature:nnnn      {CJKShape} {Simplified} {20} {1}
\@@_define_aat_feature:nnnn      {CJKShape} {JIS1978} {20} {2}
\@@_define_aat_feature:nnnn      {CJKShape} {JIS1983} {20} {3}
\@@_define_aat_feature:nnnn      {CJKShape} {JIS1990} {20} {4}
\@@_define_aat_feature:nnnn      {CJKShape} {Expert} {20} {10}
\@@_define_aat_feature:nnnn      {CJKShape} {NLC} {20} {13}
```

38.8.12 Character width

```
\@@_define_aat_feature_group:n {CharacterWidth}
\@@_define_aat_feature:nnnn      {CharacterWidth} {Proportional} {22} {0}
\@@_define_aat_feature:nnnn      {CharacterWidth} {Full} {22} {1}
\@@_define_aat_feature:nnnn      {CharacterWidth} {Half} {22} {2}
\@@_define_aat_feature:nnnn      {CharacterWidth} {Third} {22} {3}
\@@_define_aat_feature:nnnn      {CharacterWidth} {Quarter} {22} {4}
\@@_define_aat_feature:nnnn      {CharacterWidth} {AlternateProportional} {22} {5}
\@@_define_aat_feature:nnnn      {CharacterWidth} {AlternateHalf} {22} {6}
\@@_define_aat_feature:nnnn      {CharacterWidth} {Default} {22} {7}
```

38.8.13 Annotation

```
\@@_define_aat_feature_group:n {Annotation}
\@@_define_aat_feature:nnnn      {Annotation} {Off} {24} {0}
\@@_define_aat_feature:nnnn      {Annotation} {Box} {24} {1}
\@@_define_aat_feature:nnnn      {Annotation} {RoundedBox} {24} {2}
\@@_define_aat_feature:nnnn      {Annotation} {Circle} {24} {3}
\@@_define_aat_feature:nnnn      {Annotation} {BlackCircle} {24} {4}
\@@_define_aat_feature:nnnn      {Annotation} {Parenthesis} {24} {5}
\@@_define_aat_feature:nnnn      {Annotation} {Period} {24} {6}
\@@_define_aat_feature:nnnn      {Annotation} {RomanNumerals} {24} {7}
\@@_define_aat_feature:nnnn      {Annotation} {Diamond} {24} {8}
```

```
\@@_define_aat_feature:nnnn      {Annotation} {BlackSquare} {24} {9}
\@@_define_aat_feature:nnnn      {Annotation} {BlackRoundSquare} {24} {10}
\@@_define_aat_feature:nnnn      {Annotation} {DoubleCircle} {24} {11}
```

39 Extended font encodings

To be removed after the 2017 release of LaTeX2e:

```
\providecommand\UnicodeFontFile[2]{"[#1]:#2"}
\providecommand\UnicodeFontName[2]{"#1:#2"}
⟨xetexx⟩\providecommand\UnicodeFontTeXLigatures{mapping=tex-text;}
⟨luatex⟩\providecommand\UnicodeFontTeXLigatures{+tlig;}
```

```
\providecommand\add@unicode@accent[2]{#2\char#1\relax}
\providecommand\DeclareUnicodeAccent[3]{%
  \DeclareTextCommand{#1}{#2}{\add@unicode@accent{#3}}%
}
```

\EncodingCommand

```
\DeclareDocumentCommand \EncodingCommand {mO{}m}
  {
    \bool_if:NF \l_@@_defining_encoding_bool
      { \@@_error:nn {only-inside-encdef} \EncodingCommand }
    \DeclareTextCommand{#1}{\UnicodeEncodingName}[#2]{#3}
  }
```

\EncodingAccent

```
\DeclareDocumentCommand \EncodingAccent {mm}
  {
    \bool_if:NF \l_@@_defining_encoding_bool
      { \@@_error:nn {only-inside-encdef} \EncodingAccent }
    \DeclareTextCommand{#1}{\UnicodeEncodingName}{\add@unicode@accent{#2}}
  }
```

\EncodingSymbol

```
\DeclareDocumentCommand \EncodingSymbol {mm}
  {
    \bool_if:NF \l_@@_defining_encoding_bool
      { \@@_error:nn {only-inside-encdef} \EncodingSymbol }
    \DeclareTextSymbol{#1}{\UnicodeEncodingName}{#2}
  }
```

\EncodingComposite

```
\DeclareDocumentCommand \EncodingComposite {mmm}
  {
    \bool_if:NF \l_@@_defining_encoding_bool
      { \@@_error:nn {only-inside-encdef} \EncodingComposite }
    \DeclareTextComposite{#1}{\UnicodeEncodingName}{#2}{#3}
  }
```

\EncodingCompositeCommand

```
\DeclareDocumentCommand \EncodingCompositeCommand {mmm}
```

```
  {
    \bool_if:NF \l_@@_defining_encoding_bool
      { \@@_error:nn {only-inside-encdef} \EncodingCompositeCommand }
    \DeclareTextCompositeCommand{#1}{\UnicodeEncodingName}{#2}{#3}
  }
```

\DeclareUnicodeEncoding

```
\DeclareDocumentCommand \DeclareUnicodeEncoding {mm}
  {
    \DeclareFontEncoding{#1}{}{}
    \DeclareErrorFont{#1}{lmr}{m}{n}{10}
    \DeclareFontSubstitution{#1}{lmr}{m}{n}
    \DeclareFontFamily{#1}{lmr}{}

    \DeclareFontShape{#1}{lmr}{m}{n}
      {<->\UnicodeFontFile{lmroman10-regular}{\UnicodeFontTeXLigatures}}{}
    \DeclareFontShape{#1}{lmr}{m}{it}
      {<->\UnicodeFontFile{lmroman10-italic}{\UnicodeFontTeXLigatures}}{}
    \DeclareFontShape{#1}{lmr}{m}{sc}
      {<->\UnicodeFontFile{lmromancaps10-regular}{\UnicodeFontTeXLigatures}}{}
    \DeclareFontShape{#1}{lmr}{bx}{n}
      {<->\UnicodeFontFile{lmroman10-bold}{\UnicodeFontTeXLigatures}}{}
    \DeclareFontShape{#1}{lmr}{bx}{it}
      {<->\UnicodeFontFile{lmroman10-bolditalic}{\UnicodeFontTeXLigatures}}{}

    \tl_set_eq:NN \l_@@_prev_unicode_name_tl \UnicodeEncodingName
    \tl_set:Nn \UnicodeEncodingName {#1}
    \bool_set_true:N \l_@@_defining_encoding_bool
    #2
    \bool_set_false:N \l_@@_defining_encoding_bool
    \tl_set_eq:NN \UnicodeEncodingName \l_@@_prev_unicode_name_tl
  }
```

\UndeclareSymbol

```
\DeclareDocumentCommand \UndeclareSymbol {m}
  {
    \bool_if:NF \l_@@_defining_encoding_bool
      { \@@_error:nn {only-inside-encdef} \UndeclareSymbol }
    \UndeclareTextCommand {#1} {\UnicodeEncodingName}
  }

```

\UndeclareComposite

```
\DeclareDocumentCommand \UndeclareComposite {mm}
  {
    \bool_if:NF \l_@@_defining_encoding_bool
      { \@@_error:nn {only-inside-encdef} \UndeclareComposite }
    \cs_undefine:c
      { \c_backslash_str \UnicodeEncodingName \token_to_str:N #1 - \tl_to_str:n {#2} }
  }
```

40 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{arevmath}{\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:}
```

41 Closing code

41.1 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
 {
  \@@_select_font_family:nn {#1} {#2}
  \tl_set:Nn \zf@family { \l_fontspec_family_tl }
  \tl_set:Nn \zf@basefont { \l_fontspec_font }
 }
```

41.2 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.}}
 }
```

42 Patching code

⟨*fontspec⟩

42.1 Italic small caps and so on

\sishape
\textsi

These commands for actually selecting italic small caps have been defined for many years; I'm inclined to drop them. They're probably used very infrequently; I personally prefer just writing \textit{\textsc{...}} instead.

```
\providecommand*\itscdefault{\itdefault\scdefault}
\providecommand*\slscdefault{\sldefault\scdefault}
\DeclareRobustCommand{\sishape}
 {
  \not@math@alphabet\sishape\relax
  \fontshape{\itscdefault}\selectfont
 }
\DeclareTextFontCommand{\textsi}{\sishape}
```

LaTeX's 'shape' font axis needs to be overloaded to support italic small caps and slanted small caps. These are the combinations to support:

```
\cs_new:Nn \@@_shape_merge:nn { c_@@_shape_#1_#2_tl }
\tl_const:cn { \@@_shape_merge:nn \itdefault   \scdefault } {\itscdefault}
\tl_const:cn { \@@_shape_merge:nn \sldefault   \scdefault } {\slscdefault}
\tl_const:cn { \@@_shape_merge:nn \scdefault   \itdefault } {\itscdefault}
\tl_const:cn { \@@_shape_merge:nn \scdefault   \sldefault } {\slscdefault}
\tl_const:cn { \@@_shape_merge:nn \slscdefault \itdefault } {\itscdefault}
\tl_const:cn { \@@_shape_merge:nn \itscdefault \sldefault } {\slscdefault}
```

```
\tl_const:cn { \@@_shape_merge:nn \itscdefault \updefault } {\scdefault}
\tl_const:cn { \@@_shape_merge:nn \slscdefault \updefault } {\scdefault}
```

\fontspec_merge_shape:n

These macros enable the overload on the \..shape commands. First, a shape 'new+current' (prefix) or 'current+new' (suffix) is tried. If not found, fall back on the 'new' shape.

```
\cs_new:Nn \fontspec_merge_shape:n
 {
  \bool_if:nTF
    {
      \tl_if_exist_p:c { \@@_shape_merge:nn {\f@shape} {#1} }   &&
      \cs_if_exist_p:c
        {
          \f@encoding/\f@family/\f@series/
          \tl_use:c { \@@_shape_merge:nn {\f@shape} {#1} }
        }
    }
    { \fontshape { \tl_use:c { \@@_shape_merge:nn {\f@shape} {#1} } } \selectfont }
    { \fontshape {#1} \selectfont }
 }
```

\itshape
\scshape
\upshape
\slshape

The original \..shape commands are redefined to use the merge shape macro.

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

42.2 Emphasis

\em
\emph
\emshape
\eminnershape
\emfontdeclare

Redefinition of {\em ...} and \emph{...} to allow nesting of emphases.

```
\cs_new_protected:Npn \emfontdeclare #1
  {
    \prop_clear:N \g_@@_em_prop
    \int_zero:N   \l_@@_emdef_int
    \group_begin:
      \normalfont
      \clist_map_inline:nn {\emreset,#1}
        {
          ##1
          \prop_gput:NxV \g_@@_em_prop { \f@series/\f@shape } { \l_@@_emdef_int }
          \prop_gput:Nxn \g_@@_em_prop { switch-\int_use:N \l_@@_emdef_int } { ##1 }
          \int_incr:N \l_@@_emdef_int
        }
    \group_end:
  }
\DeclareRobustCommand \em
  {
    \@nomath\em

    \prop_get:NxNT \g_@@_em_prop { \f@series/\f@shape } \l_@@_em_tmp_tl
      { \int_set:Nn \l_@@_em_int { \l_@@_em_tmp_tl } }

    \int_incr:N \l_@@_em_int

    \prop_get:NxNTF \g_@@_em_prop { switch-\int_use:N \l_@@_em_int } \l_@@_em_switch_tl
      { \l_@@_em_switch_tl }
      {
        \int_zero:N \l_@@_em_int
        \emreset
      }

  }
```

Document commands:

```
\DeclareTextFontCommand{\emph}{\em}
\cs_set:Npn \emreset { \upshape }
\cs_set:Npn \emshape { \itshape }
\cs_set:Npn \eminnershape { \upshape }
\emfontdeclare{ \emshape, \eminnershape }
```

Ensure nesting resets when necessary:

```
\cs_set:Npn \reset@font { \normalfont \int_zero:N \l__fontspec_em_int }
```

Programmer's interface for setting nesting level:

```
\cs_new:Nn \fontspec_set_em_level:n { \int_set:Nn \l_@@_em_int {#1} }
```

42.3 \-

\- 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{`\-}
```

42.4 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+2423: OPEN BOX, which is used to visibly display a space character.

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

`tspec_visible_space_fallback:` If the current font doesn't have U+2423: 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
  }
 }
```

`ontspec_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:
  }{}
}
```

42.5 \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 }
 }
```

```
⟨/fontspec⟩
```

Index

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