

The fontspec package

Font selection for X_YLaTeX and LuaLaTeX

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

? ?

Contents

I	Getting started	4
1	History	4
2	Introduction	4
2.1	Acknowledgements	4
3	Package loading and options	5
3.1	Maths fonts adjustments	5
3.2	Configuration	6
3.3	Warnings	6
II	General font selection	7
4	Font selection	7
4.1	By font name	8
4.2	By file name	8
4.3	Querying whether a font ‘exists’	9
5	Commands to select font families	10
5.1	More control over font shape selection	11
5.2	Specifically choosing the NFSS family	12
5.3	Choosing additional NFSS font faces	13
5.4	Math(s) fonts	14
6	Miscellaneous font selecting details	15
III	OpenType	17
7	Introduction	17

7.1	How to select font features	17
7.2	How do I know what font features are supported by my fonts?	18
8	OpenType font features	19
8.1	Tag-based features	19
8.2	Letters	19
8.3	Style	26
8.4	Diacritics	28
8.5	Kerning	28
8.6	Character width	28
8.7	Vertical typesetting	30
8.8	Numeric features	32
8.9	OpenType scripts and languages	35
IV	Commands for accents and symbols ('encodings')	38
9	A new Unicode-based encoding from scratch	38
10	Adjusting a pre-existing encoding	39
11	Summary of commands	41
V	LuaTeX-only font features	42
12	Custom font features	42
VI	Fonts and features with XeTeX	43
13	XeTeX-only font features	43
13.1	Mapping	43
13.2	Different font technologies: AAT and OpenType	43
13.3	Optical font sizes	43
14	Mac OS X's AAT fonts	44
14.1	Ligatures	44
14.2	Letters	44
14.3	Numbers	44
14.4	Contextuals	45
14.5	Vertical position	45
14.6	Fractions	46
14.7	Variants	46
14.8	Alternates	46
14.9	Style	47
14.10	CJK shape	47
14.11	Character width	47
14.12	Vertical typesetting	47

14.13	Diacritics	48
14.14	Annotation	48
VII	Customisation and programming interface	49
15	Defining new features	49
16	Defining new scripts and languages	50
17	Going behind fontspec's back	50
18	Renaming existing features & options	50
19	Programming interface	51
19.1	Variables	51
19.2	Functions for loading new fonts and families	51
19.3	Conditionals	52
VIII	The 'improvement' of L^AT_EX 2_ε and other packages	54
20	Verbatim	54
21	Discretionary hyphenation: \-	54
22	Commands for old-style and lining numbers	54

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 X_{\LaTeX} , the first widely-used Unicode extension to \TeX . Over time, X_{\LaTeX} was extended to support OpenType fonts and then was ported into a cross-platform program to run also on Windows and Linux.

More recently, $\text{Lua}\TeX$ is fast becoming the \TeX engine of the day; it supports Unicode encodings and OpenType fonts and opens up the internals of \TeX via the Lua programming language. Hans Hagen's $\text{Con}\TeX\text{t Mk. IV}$ is a re-write of his powerful typesetting system, taking full advantage of $\text{Lua}\TeX$'s features including font support; a kernel of his work in this area has been extracted to be useful for other \TeX macro systems as well, and this has enabled fontspec to be adapted for \LaTeX when run with the $\text{Lua}\TeX$ engine.

2 Introduction

The fontspec package allows users of either X_{\LaTeX} or $\text{Lua}\TeX$ to load OpenType fonts in a \LaTeX document. No font installation is necessary, and font features can be selected and used as desired throughout the document.

Without fontspec, it is necessary to write cumbersome font definition files for \LaTeX , since \LaTeX 's font selection scheme (known as the 'nfss') has a lot going on behind the scenes to allow easy commands like `\emph` or `\bfseries`. With an uncountable number of fonts now available for use, however, it becomes less desirable to have to write these font definition (`.fd`) files for every font one wishes to use.

Because fontspec is designed to work in a variety of modes, this user documentation is split into separate sections that are designed to be relatively independent. Nonetheless, the basic functionality all behaves in the same way, so previous users of fontspec under X_{\LaTeX} should have little or no difficulty switching over to $\text{Lua}\TeX$.

This manual can get rather in-depth, as there are a lot of details to cover. See the 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 X_{\LaTeX} , 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 Lua \TeX , 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.

UPDATE!

Font encodings The 2016 release of fontspec initiated some changes for font encodings and the loading of xunicode.

A new package option, `tuenc`, which is selected by default, switches the NFSS font encoding to TU. TU is a new Unicode font encoding, intended for both X \TeX and Lua \TeX engines, and automatically contains support for symbols covered by \TeX '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.

The old behaviour can be achieved by loading the `euenc` package option. This selects the EU1 or EU2 encoding (X \TeX /Lua \TeX , 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 19 on page 51](#) for how to do this properly.)

Lua \TeX users only In order to load fonts by their name rather than by their file-name (e.g., 'Latin Modern Roman' instead of 'ec-lmr10'), you may need to run the script `luaotfload-tool`, which is distributed with the luaotfload package. Note that if you do not execute this script beforehand, the first time you attempt to typeset the process will pause for (up to) several minutes. (But only the first time.) Please see the luaotfload documentation for more information.

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

3.1 Maths fonts adjustments

By default, fontspec adjusts \TeX '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 selection of the maths fonts.

3.2 Configuration

If you wish to customise any part of the `fontspec` interface, this should be done by creating your own `fontspec.cfg` file, which will be automatically loaded if it is found by Xe_{La}TeX or Lua_{La}TeX. 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 Xe_{La}TeX or Lua_{La}TeX searches by default; *e.g.* in Mac_{La}TeX: `~/Library/texmf/tex/latex/`.

The package option `no-config` will suppress the loading of the `fontspec.cfg` file under all circumstances.

3.3 Warnings

This package can give 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.

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

	<code>\setmainfont{texgyrebonum-regular.otf}</code>
	<code>\setsansfont{lmsansio-regular.otf}[Scale=MatchLowercase]</code>
	<code>\setmonofont{Inconsolata.otf}[Scale=MatchLowercase]</code>
Pack my box with five dozen liquor jugs	<code>\rmfamily Pack my box with five dozen liquor jugs\par</code>
Pack my box with five dozen liquor jugs	<code>\sffamily Pack my box with five dozen liquor jugs\par</code>
Pack my box with five dozen liquor jugs	<code>\ttfamily Pack my box with five dozen liquor jugs</code>

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 ?? on page ??, 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 ?? on page ??.

The font features argument accepts comma separated `=<option>` lists; these are described in later:

- For general font features, see ?? on page ??
- For OpenType fonts, see Part III on page 17
- For X_YTeX-only general font features, see Part VI on page 43
- For LuaTeX-only general font features, see Part V on page 42
- For features for AAT fonts in X_YTeX, see Section 14 on page 44

4 Font selection

In both LuaTeX and X_YTeX, 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.

4.1 By font name

Fonts known to Lua \TeX or X \TeX 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 Lua \TeX , 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, \TeX Live contains the `otfinfo` command line program, which can query this information; for example:

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

results in 'LM Roman 10'.

4.2 By file name

X \TeX and Lua \TeX 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 X \TeX 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 X \TeX but cannot be loaded by font name, only file name; Lua \TeX 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 X_YTeX and LuaTeX are able to load the font without giving an extension, but fontspec must know to search for the file; this can be indicated by 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.

4.3 Querying whether a font ‘exists’

```

\IfFontExistsTF{<font name>}{<true branch>}{<>false branch>}

```

The conditional \IfFontExistsTF is provided to test whether the 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 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}

```

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

```
\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 19 on page 51](#)).

5 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 14.4 on page 45](#).

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 4: Explicit selection of the bold font.

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

```

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

```

5.1 More control over font shape selection

```

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

```

The automatic bold, italic, and bold italic font selections will not be adequate for the needs of every font: while some fonts mayn't even have bold or italic shapes, in which case a skilled (or lucky) designer may be able to chose well-matching accompanying shapes from a different font altogether, others can have a range of bold and italic fonts to chose among. The `BoldFont` and `ItalicFont` features are provided for these situations. If only one of these is used, the bold italic font is requested as the default from the new font. See Example 4.

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

5.1.1 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 \TeX 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 ?? 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},
]
```

5.2 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

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

```
\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 original 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 19 on page 51](#)).

5.3 Choosing additional NFSS font faces

TeX'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:

```
\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 $\text{\LaTeX 2}\varepsilon$ font selection guide² lists series m for medium, b for bold, bx for bold extended, sb for semi-bold, and c for condensed. A far more comprehensive listing is included in Appendix A of Philipp Lehman's 'The Font Installation Guide'³ covering 14 separate weights and 12 separate widths.

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

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

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

5.4 Math(s) fonts

When `\setmainfont`, `\setsansfont` and `\setmonofont` are used in the preamble, they also define the fonts to be used in maths mode inside the `\mathrm`-type commands. This only occurs in the preamble because \LaTeX freezes the maths fonts after this stage of the processing. The `fontspec` package must also be loaded after any maths font packages (*e.g.*, `euler`) to be successful. (Actually, it is only `euler` that is the problem.⁴)

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

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

```
\setmathrm{<font name>}[<font features>]
\setmathsf{<font name>}[<font features>]
\setmathtt{<font name>}[<font features>]
\setboldmathrm{<font name>}[<font features>]
```

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

²`texdoc fntguide`

³`texdoc fontinstallationguide`

⁴Speaking 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}}`

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.

6 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 as 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.

Italic small caps 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 `\slshape`. In addition, once italic small caps are selected then `\slshape` will switch to slanted small caps, and vice versa.)

Emphasis and nested emphasis $\text{\LaTeX 2}_{\epsilon}$ allows you to specify the behaviour of `\emph` nested within `\emph` by setting the `\emminnershape` command. For example, `\renewcommand\emminnershape{\upshape\scshape}` will produce small caps within `\emph{\emph{...}}`.

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 features 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.

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 III

OpenType

7 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 8](#).

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.

7.1 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 [8.1.1](#).

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

If a feature or an option is requested that the font does not have, a warning is given in the console output. As mentioned in [Section 3.3 on page 6](#) these warnings can be

suppressed by selecting the [quiet] package option.

7.2 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 T_EX editor and change the font name to the font you'd like to query; after running through plain X_YT_EX, 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 T_EXLive. Simply type in a Terminal window, say:

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

which results in:

aalt	Access All Alternates
csp	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

8 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 1 on the following 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 2 on page 21](#). 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.

8.1 Tag-based features

8.1.1 Ligatures

Ligatures refer to the replacement of two separate characters with a specially drawn glyph for functional or aesthetic reasons. The list of options, of which multiple may be selected at one time, is shown in [Table 3](#). A demonstration with the Linux Libertine fonts⁵ is shown in [Example 5](#).

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 13.1 on page 43](#)) but for consistency `Ligatures=TeX` will perform the same function as `Mapping=tex-text`.

8.2 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`.

⁵<http://www.linuxlibertine.org/>

Table 1: Summary of OpenType features in fontspec, alphabetic by feature tag.

ABVM	Diacritics = AboveBase	<i>Above-base Mark Positioning</i>	NUMR	VerticalPosition = Numerator	<i>Numerators</i>
AFRC	Fractions = Alternate	<i>Alternative Fractions</i>	ONUM	Numbers = Lowercase	<i>Oldstyle Figures</i>
BLWM	Diacritics = BelowBase	<i>Below-base Mark Positioning</i>	ORDN	VerticalPosition = Ordinal	<i>Ordinals</i>
			ORNM	Ornament = <i>N</i>	<i>Ornaments</i>
CALT	Contextuals = Alternate	<i>Contextual Alternates</i>	PALT	CharacterWidth = AlternateProportional	<i>Proportional Alternate Widths</i>
CASE	Letters = Uppercase	<i>Case-Sensitive Forms</i>	PCAP	Letters = PetiteCaps	<i>Petite Capitals</i>
CLIG	Ligatures = Contextual	<i>Contextual Ligatures</i>	PKNA	Style = ProportionalKana	<i>Proportional Kana</i>
CPSP	Kerning = Uppercase	<i>Capital Spacing</i>	PNUM	Numbers = Proportional	<i>Proportional Figures</i>
CSWH	Contextuals = Swash	<i>Contextual Swash</i>	PWID	CharacterWidth = Proportional	<i>Proportional Widths</i>
CVNN	CharacterVariant = <i>N:M</i>	<i>Character Variant N</i>	QWID	CharacterWidth = Quarter	<i>Quarter Widths</i>
C2PC	Letters = UppercasePetiteCaps	<i>Petite Capitals From Capitals</i>	RAND	Letters = Random	<i>Randomize</i>
			RLIG	Ligatures = Required	<i>Required Ligatures</i>
C2SC	Letters = UppercaseSmallCaps	<i>Small Capitals From Capitals</i>	RUBY	Style = Ruby	<i>Ruby Notation Forms</i>
			SALT	Alternate = <i>N</i>	<i>Stylistic Alternates</i>
DLIG	Ligatures = Rare	<i>Discretionary Ligatures</i>	SINF	VerticalPosition = ScientificInferior	<i>Scientific Inferiors</i>
DNOM	VerticalPosition = Denominator	<i>Denominators</i>	SMCP	Letters = SmallCaps	<i>Small Capitals</i>
EXPT	CJKShape = Expert	<i>Expert Forms</i>	SMPL	CJKShape = Simplified	<i>Simplified Forms</i>
FALT	Contextuals = LineFinal	<i>Final Glyph on Line Alternates</i>	ssNN	StylisticSet = <i>N</i>	<i>Stylistic Set N</i>
			SSTY	Style = MathScript	<i>Math script style alternates</i>
FINA	Contextuals = WordFinal	<i>Terminal Forms</i>	SUBS	VerticalPosition = Inferior	<i>Subscript</i>
FRAC	Fractions = On	<i>Fractions</i>	SUPS	VerticalPosition = Superior	<i>Superscript</i>
FWID	CharacterWidth = Full	<i>Full Widths</i>	SWSH	Style = Swash	<i>Swash</i>
HALT	CharacterWidth = AlternateHalf	<i>Alternate Half Widths</i>	TITL	Style = TitlingCaps	<i>Titling</i>
HIST	Style = Historic	<i>Historical Forms</i>	TNUM	Numbers = Monospaced	<i>Tabular Figures</i>
HKNA	Style = HorizontalKana	<i>Horizontal Kana Alternates</i>	TRAD	CJKShape = Traditional	<i>Traditional Forms</i>
HLIG	Ligatures = Historic	<i>Historical Ligatures</i>	TWID	CharacterWidth = Third	<i>Third Widths</i>
HWID	CharacterWidth = Half	<i>Half Widths</i>	UNIC	Letters = Unicae	<i>Unicae</i>
INIT	Contextuals = WordInitial	<i>Initial Forms</i>	VALT	Vertical = AlternateMetrics	<i>Alternate Vertical Metrics</i>
ITAL	Style = Italic	<i>Italics</i>	VERT	Vertical = Alternates	<i>Vertical Writing</i>
JP78	CJKShape = JIS1978	<i>JIS78 Forms</i>	VHAL	Vertical = HalfMetrics	<i>Alternate Vertical Half Metrics</i>
JP83	CJKShape = JIS1983	<i>JIS83 Forms</i>			
JP90	CJKShape = JIS1990	<i>JIS90 Forms</i>	VKNA	Style = VerticalKana	<i>Vertical Kana Alternates</i>
JP04	CJKShape = JIS2004	<i>JIS2004 Forms</i>	VKRN	Vertical = Kerning	<i>Vertical Kerning</i>
KERN	Kerning = On	<i>Kerning</i>	VPAL	Vertical = ProportionalMetrics	<i>Proportional Alternate Vertical Metrics</i>
LIGA	Ligatures = Common	<i>Standard Ligatures</i>			
LNUM	Numbers = Uppercase	<i>Lining Figures</i>	VRT2	Vertical = RotatedGlyphs	<i>Vertical Alternates and Rotation</i>
MARK	Diacritics = MarkToBase	<i>Mark Positioning</i>			
MEDI	Contextuals = Inner	<i>Medial Forms</i>	VRTR	Vertical = AlternatesForRotation	<i>Vertical Alternates for Rotation</i>
MKMK	Diacritics = MarkToMark	<i>Mark to Mark Positioning</i>			
NALT	Annotation = <i>N</i>	<i>Alternate Annotation Forms</i>	ZERO	Numbers = SlashedZero	<i>Slashed Zero</i>
NLCK	CJKShape = NLC	<i>NLC Kanji Forms</i>			

Table 2: List of unsupported OpenType features.

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

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

Feature	Option	Tag
Ligatures =	Required	r _{lig} †
	Common	l _{iga} †
	Contextual	c _{lig} †
	Rare/Discretionary	d _{lig} †
	Historic	h _{lig} †
	TeX	t _{lig} †
ResetAll		

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

Example 5: An example of the Ligatures feature.

strict	→	ſtrict	<pre> \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} </pre>
wurtzite	→	wurtzite	
firefly	→	firefly	

Table 4: 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`.

Example 6: Small caps from lowercase or uppercase letters.

	<code>\fontspec{texgyreadventor-regular.otf}[Letters=SmallCaps]</code>
THIS SENTENCE NO VERB	THIS SENTENCE no verb
	<code>\fontspec{texgyreadventor-regular.otf}[Letters=UppercaseSmallCaps]</code>
THIS SENTENCE NO verb	THIS SENTENCE no verb

Example 7: An example of the Uppercase option of the Letters feature.

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

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 6. ‘Unicase’ is a weird hybrid of upper and lower case letters.

Note that the `Uppercase` option will (probably) not actually map letters to uppercase.⁶ It is designed to select various uppercase forms for glyphs such as accents and dashes, such as shown in Example 7; 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 8.5 on page 28](#)).

8.2.1 Numbers

The `Numbers` feature defines how numbers will look in the selected font, accepting options shown in [Table 5](#).

The synonyms `Uppercase` and `Lowercase` are equivalent to `Lining` and `Old-Style`, respectively. The differences have been shown previously in ?? on page ??. 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 8.

The `Arabic` option (with tag `anum`) maps regular numerals to their Arabic script or Persian equivalents based on the current `Language` setting (see [Section 8.9 on page 35](#)). This option is based on a LuaTeX feature of the `luaotfload` package, not an OpenType feature. (Thus, this feature is unavailable in XeTeX.)

⁶If you want automatic uppercase letters, look to LaTeX’s `\MakeUppercase` command.

Table 5: 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 8: The effect of the SlashedZero option.

	<code>\fontspec[Numbers=Lining]{texgyrebonum-regular.otf}</code>
	0123456789
0123456789 0123456789	<code>\fontspec[Numbers=SlashedZero]{texgyrebonum-regular.otf}</code>
	0123456789

8.2.2 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 6](#).

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.

8.2.3 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 9](#)

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

The `realscripts` package (which is also loaded by `xltxtra` for \LaTeX) 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.

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

Feature	Option	Tag
Contextuals =	Swash	cswh †
	Alternate	calt †
	WordInitial	init †
	WordFinal	fini †
	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 7: Options for the OpenType font feature ‘VerticalPosition’.

Feature	Option	Tag
VerticalPosition =	Superior	supr †
	Inferior	subr †
	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.

Example 9: The VerticalPosition feature.

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

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

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

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

Example 10: The Fractions feature.

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

8.2.4 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 10.

8.3 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 11 and Example 12 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 13 and Example 14; in the latter, the Italic option affects the Latin text and the Ruby option the Japanese.

Example 11: Example of the Alternate option of the Style feature.

M Q W	<code>\fontspec{Quattrocento Roman}</code>
M Q W	<code>M Q W \quad \backslash</code>
M Q W	<code>\addfontfeature{Style=Alternate}</code>
M Q W	<code>M Q W</code>

Table 9: 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 12: 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 13: 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 14: 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 15: Example of the HorizontalKana and VerticalKana options of the Style feature.

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

Table 10: 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`.

Note the difference here between the default and the horizontal style kana in Example 15: the horizontal style is slightly wider.

8.4 Diacritics

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

8.5 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 8.2 on page 19, the Uppercase option will add a small amount of tracking between uppercase letters, seen in Example 16, which uses the Romande fonts⁷ (thanks to Clea F. Rees for the suggestion). The Uppercase option acts separately to the regular kerning controlled by the On/Off options.

8.6 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.

⁷<http://arkandis.tuxfamily.org/adffonts.html>

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

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

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

Example 16: 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 12: Options for the OpenType font feature ‘CharacterWidth’.

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

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

Example 17: Proportional or fixed width forms.

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

Example 18: Numbers can be compressed significantly.

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

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 18.

8.6.1 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 glyphs correctly in whichever form is appropriate. Both AAT and OpenType fonts support the following CJKShape options: Traditional, Simplified, JIS1978, JIS1983, JIS1990, and Expert. OpenType also supports the NLC option.

8.7 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.

Table 13: Options for the OpenType font feature ‘CJKShape’.

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

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

Example 19: Different standards for CJK ideograph presentation.

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

Table 14: Options for the OpenType font feature ‘Vertical’.

Feature	Option	Tag
Vertical =	RotatedGlyphs	vrt2 †
	AlternatesForRotation	vrtr †
	Alternates	vert †
	KanaAlternates	vkna †
	Kerning	vkrr †
	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 20: Insular letterforms, as used in medieval Northern Europe, for the Junicode font accessed with the `StylisticSet` feature.

Insular forms.	<code>\fontspec{Junicode}</code>
Insular forms.	<code>Insular forms. \</code>
Insular forms.	<code>\addfontfeature{StylisticSet=2}</code>
Insular forms.	<code>Insular forms. \</code>

Example 21: Enlarged minuscules (capital letters remain unchanged) for the Junicode font, accessed with the `StylisticSet` feature.

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

8.8 Numeric features

8.8.1 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 font⁸ are shown in Example 20 and Example 21; 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 15 on page 49 for a way to assign names to stylistic sets, which should be done on a per-font basis.

8.8.2 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 `cvo1` to `cv99`.

For each character that can be varied, it is possible to select among possible options for that particular glyph. For example, in Example 22 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 `:<n>` corresponds to which variety to choose. Georg Duffner’s open source Garamond revival font⁹ 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 23. (Unlike stylistic alternates, say.)

Note that the indexing starts from zero.

⁸<http://junicode.sf.net>

⁹<http://www.georgduffner.at/ebgaramond/>

Example 22: The CharacterVariant feature showing off Georg Duffner’s open source Garamond revival font.

<i>very</i>	
<i>very</i>	
<i>very</i>	
<i>very</i>	<code>\fontspec{EB Garamond 12 Italic} very \\</code>
<i>very</i>	<code>\fontspec{EB Garamond 12 Italic}[CharacterVariant=5] very \\</code>
<i>very</i>	<code>\fontspec{EB Garamond 12 Italic}[CharacterVariant=5:0] very \\</code>
<i>very</i>	<code>\fontspec{EB Garamond 12 Italic}[CharacterVariant=5:1] very \\</code>
<i>very</i>	<code>\fontspec{EB Garamond 12 Italic}[CharacterVariant=5:2] very \\</code>
<i>very</i>	<code>\fontspec{EB Garamond 12 Italic}[CharacterVariant=5:3] very</code>

Example 23: The CharacterVariant feature selecting multiple variants simultaneously.

<i>ℰ violet</i>	
<i>ℰ violet</i>	
<i>ℰ violet</i>	<code>\fontspec{EB Garamond 12 Italic} \& violet \\</code>
<i>ℰ violet</i>	<code>\fontspec{EB Garamond 12 Italic}[CharacterVariant={4}] \& violet \\</code>
<i>ℰ violet</i>	<code>\fontspec{EB Garamond 12 Italic}[CharacterVariant={5:2}] \& violet \\</code>
<i>ℰ violet</i>	<code>\fontspec{EB Garamond 12 Italic}[CharacterVariant={4,5:2}] \& violet</code>

Example 24: The Alternate feature.	
a & h	<code>\fontspec{Linux Libertine 0}</code>
a & h	<code>\textsc{a} \& h \</code>
a & h	<code>\addfontfeature{Alternate=o}</code>
	<code>\textsc{a} \& h</code>

Example 25: Annotation forms for OpenType fonts.	
1 2 3 4 5 6 7 8 9	
1 2 3 4 5 6 7 8 9	
1 2 3 4 5 6 7 8 9	
1 2 3 4 5 6 7 8 9	
1 2 3 4 5 6 7 8 9	
1 2 3 4 5 6 7 8 9	
1 2 3 4 5 6 7 8 9	
1 2 3 4 5 6 7 8 9	
1 2 3 4 5 6 7 8 9	<code>\fontspec{Hiragino Maru Gothic Pro}</code>
1 2 3 4 5 6 7 8 9	<code>1 2 3 4 5 6 7 8 9</code>
1 2 3 4 5 6 7 8 9	<code>\def\x#1{\{\addfontfeature{Annotation=#1}</code>
1 2 3 4 5 6 7 8 9	<code>1 2 3 4 5 6 7 8 9 \}}</code>
1 2 3 4 5 6 7 8 9	<code>\xo\x1\x2\x3\x4\x5\x6\x7\x8\x9</code>

8.8.3 Alternates — salt

The Alternate feature, alias StylisticAlternates, is used to access alternate font glyphs when variations exist in the font, such as in Example 24. 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=o` to access the default case.

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

See [Section 15 on page 49](#) for a way to assign names to alternates if desired.

8.8.4 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 25. Note that the indexing starts from zero.

8.8.5 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.

8.9 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 15 on the next page](#) and [Table 16 on page 37](#), respectively. When a script or language is requested that is not supported by the current font, a warning is printed in the console output.

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 X_YTeX is being used, will specifically select the OpenType renderer for this font, as described in [Section 13.2 on page 43](#).

See [Section 16 on page 50](#) for methods to create new Script or Language options if required.

8.9.1 Script and Language examples

In the examples shown in [Example 26](#), the Code2000 font¹⁰ 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.

¹⁰<http://www.code2000.net/>

Example 26: An example of various Scripts and Languages.

العربي	العربي	
हिन्दी	हिन्दी	
লেখ	লেখ	
ਮੁੱਢਲਾ-ਸ਼ੁੱਧ ਨਿਵੇਦਨ	ਮੁੱਢਲਾ-ਸ਼ੁੱਧ ਨਿਵੇਦਨ	<code>\testfeature{Script=Arabic}{\arabictext}</code>
നമുടാ പാരബരയ	നമുടാ പാരബരയ	<code>\testfeature{Script=Devanagari}{\devanagaritext}</code>
আদি সচু জগাচি সচু	আদি সচু জগাচি সচু	<code>\testfeature{Script=Bengali}{\bengalitext}</code>
தமிழ் துடே	தமிழ் துடே	<code>\testfeature{Script=Gujarati}{\gujaratitext}</code>
תנ"ך. תנ"ך	תנ"ך. תנ"ך	<code>\testfeature{Script=Malayalam}{\malayalamtext}</code>
cáp số mõi	cáp số mõi	<code>\testfeature{Script=Gurmukhi}{\gurmukhitext}</code>
		<code>\testfeature{Script=Tamil}{\tamiltext}</code>
		<code>\testfeature{Script=Hebrew}{\hebrewtext}</code>
		<code>\def\examplefont{Doulos SIL}</code>
		<code>\testfeature{Language=Vietnamese}{\vietnamesetext}</code>

Table 15: Defined Scripts for OpenType fonts. Aliased names are shown in adjacent positions marked with red pilcrow (¶).

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

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

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

Part IV

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 $\text{\LaTeX}2_{\epsilon}$ 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 $\text{\LaTeX}2_{\epsilon}$ 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 $\text{\LaTeX}2_{\epsilon}$ input methods, accents and symbols were input using encoding-dependent commands such as `\`e` as described above. As of 2017, in $\text{\LaTeX}2_{\epsilon}$ 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.

9 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:

```

57 \DeclareTextSymbol{\CYRIE}{OT2}{5}
58 \DeclareTextSymbol{\CYRDJE}{OT2}{6}
59 \DeclareTextSymbol{\CYRTSHE}{OT2}{7}
60 \DeclareTextSymbol{\cyrnje}{OT2}{8}
61 \DeclareTextSymbol{\cyr1je}{OT2}{9}
62 \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{\cyr1je} {\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.

10 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}{
  \ImportEncoding{TU}
  \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 are loading a font that you know is missing, say, the interrobang (not that unusual a situation), you might write:

```
\DeclareUnicodeEncoding{nobang}{
  \ImportEncoding{TU}
  \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 in 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}{
  \ImportEncoding{TU}
  \EncodingCommand{\d}[1]{%
    \hmode\bgroup
      \o@lign{\relax#1\cr\hidewidth\ltx@sh@ft{-1ex}.\hidewidth}%
    \egroup
  }
}
```

This would be set up in a document as such:

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

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

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

11 Summary of commands

The \LaTeX 2 ϵ 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 are provided by `fontspec` to simplify the process of defining Unicode font range encodings:

```
\ImportEncoding{TU}
\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`.

Figure 1: An example of custom font features.

```
\documentclass{article}
\usepackage{fontspec}
\directlua{
  fonts.handlers.otf.addfeature {
    name = "oneb",
    {
      type = "substitution",
      data = {
        ["1"] = "one.sso1",
      },
    },
    "feature oneb for vollkorn font",
  }
}
\setmainfont{Vollkorn-Regular.otf}[RawFeature=+oneb]
\begin{document}
1234567890
\end{document}
```

Part V

LuaTeX-only font features

12 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/luatex 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/luatex documentation for more details.

Example 27: X_YTeX's Mapping feature.

<code>"¡A small amount of—text!"</code>	<code>\fontspec{Cochin}[Mapping=tex-text]</code>
	<code>``!`A small amount of---text!''</code>

Part VI

Fonts and features with X_YTeX

13 X_YTeX-only font features

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

13.1 Mapping

Mapping enables a X_YTeX text-mapping scheme, shown in Example 27.

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

13.2 Different font technologies: AAT and OpenType

X_YTeX supports two rendering technologies for typesetting, selected with the `Renderer` font feature. The first, AAT, is that provided (only) by Mac OS X itself. The second, OpenType, is an open source OpenType interpreter.¹¹ 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 8.9 on page 35](#) 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.

13.3 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.

¹¹v2.4: This was called 'ICU' in previous versions of X_YTeX and `fontspec`. Backwards compatibility is preserved.

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          \\\
```

14 Mac OS X's AAT fonts

Warning! $X_{\text{\LaTeX}}$'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 $X_{\text{\LaTeX}}$'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 $X_{\text{\LaTeX}}$ (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'.

14.1 Ligatures

Ligatures refer to the replacement of two separate characters with a specially drawn glyph for functional or aesthetic 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 $X_{\text{\LaTeX}}$; you must use a `opt 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').

14.2 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.

14.3 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

Example 28: Contextual glyph for the beginnings and ends of words.

<p>[Contextuals=WordInitial,WordFinal] <i>where is all the veg-</i> <i>emite</i></p>	<pre>\newfontface\fancy{Hoefler Text Italic} [Contextuals={WordInitial,WordFinal}] \fancy where is all the vegemite</pre>
--	---

Example 29: A contextual feature for the ‘long s’ can be convenient as the character does not need to be marked up explicitly.

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

(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 ?? on page ??.

14.4 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 28), LineInitial, LineFinal, and Inner (Example 29, also called ‘non-final’ sometimes). As non-exclusive selectors, like the ligatures, you can turn them off by prefixing their name with No.

14.5 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 30.

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

Example 30: Vertical position for AAT fonts.

	<pre>\fontspec{Skia} Normal \fontspec{Skia}[VerticalPosition=Superior] Superior \fontspec{Skia}[VerticalPosition=Inferior] Inferior \fontspec{Skia}[VerticalPosition=Ordinal] 1st 2nd 3rd 4th 0th 8abcde</pre>
<p>Normal ^{superior} _{inferior} 1st 2nd 3rd 4th 0th 8abcde</p>	

Example 31: Fractions in AAT fonts. The `~~~~2044` glyph is the ‘fraction slash’ that may be typed in Mac OS X with `OPT+SHIFT+1`; not shown literally here due to font constraints.

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

Example 32: Alternate design of pre-composed fractions.

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

footnote labels.

14.6 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 31.

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 32.

14.7 Variants


The `Variant` feature takes a single numerical input for choosing different alphabetic shapes. Don’t mind my fancy Example 33 :) I’m just looping through the nine (!) variants of Zapfino.

See [Section 15 on page 49](#) for a way to assign names to variants, which should be done on a per-font basis.

14.8 Alternates

Selection of Alternates again must be done numerically; see Example 34. See [Section 15 on page 49](#) for a way to assign names to alternates, which should be done on a per-font basis.

Example 33: Nine variants of Zapfino.

	<pre> \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} </pre>
---	---

Example 34: Alternate shape selection must be numerical.

<p><i>Sphinx Of Black Quartz, JUDGE My Vow</i></p> <p><i>Sphinx Of Black Quartz, JUDGE Mr Vow</i></p>	<pre> \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} </pre>
---	--

14.9 Style

The options of the Style feature are defined in AAT as one of the following: Display, Engraved, IlluminatedCaps, Italic, Ruby,¹² TallCaps, or TitlingCaps.

Typical examples for these features are shown in [Section 8.3](#).

14.10 CJK shape

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

14.11 Character width

See [Section 8.6 on page 28](#) 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.

14.12 Vertical typesetting

TODO: improve!

X_YTeX provides for vertical typesetting simply with the ability to rotate the individual glyphs as a font is used for typesetting, as shown in [Example 35](#).

¹²‘Ruby’ refers to a small optical size, used in Japanese typography for annotations.

共產主義者は

共產主義者は

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

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

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

14.13 Diacritics

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

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

14.14 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.

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

Part VII

Customisation and programming interface

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

15 Defining new features

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

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

`\newAATfeature{<feature>}{<option>}{<feature code>}{<selector code>}`

Use the X_YTeX file `AAT-info.tex` to obtain the code numbers. See Example 36.

`\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 X_YTeX feature that fontspec hasn't been updated to support), a command is provided to pass arbitrary input into the font selection string:

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

For example, Zapfino 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 37: Using raw font features directly.

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

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

16 Defining new scripts and languages

`\newfontscript` While the scripts and languages listed in [Table 15](#) and [Table 16](#) are intended to be comprehensive, there may be some missing; alternatively, you might wish to use different names to access scripts/languages that are already listed. Adding scripts and languages can be performed with the `\newfontscript` and `\newfontlanguage` commands. For example,

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

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

17 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]
```

18 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 38](#).

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

Example 39: Renaming font feature options.	
	<code>\aliasfontfeature{VerticalPosition}{Vert Pos}</code>
	<code>\aliasfontfeatureoption{VerticalPosition}{ScientificInferior}{Sci Inf}</code>
	<code>\fontspec{LinLibertine_R.otf}[Vert Pos=Sci Inf]</code>
Scientific Inferior: 12345	Scientific Inferior: 12345

with the command `\aliasfontfeatureoption{font feature}{existing name}{new name}`, such as shown in Example 39.

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.

19 Programming interface

19.1 Variables

`\l_fontspec_family_tl`
`\l_fontspec_font`

`\g_fontspec_encoding_tl`

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`.

Package authors who need to load fonts with legacy \LaTeX `\fontspec` 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.

19.2 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 L^AT_EX commands `\fontfamily{family}\selectfont`. See the standard fontspec user commands for applications of this function.

`\fontspec_set_fontface:NNnn`

#1 : primitive font

#2 : L^AT_EX family

#3 : fontspec features

#4 : font name

Variant of the above in which the primitive T_EX 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 L^AT_EX programmers who need to perform subsequent font-related tests on the *primitive font*.

19.3 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.

19.3.1 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 Lua_T_EX fonts.

19.3.2 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.

<code>\fontspec_if_feature:nnnTF</code>	Test whether the currently selected font with raw OpenType script tag #1 and raw OpenType language tag #2 contains the raw OpenType feature tag #3. E.g.: <code>\fontspec_if_feature:nnnTF {serif} {latn} {smallcaps}</code> . Returns false if the font is not loaded by fontspec or is not an OpenType font.
<code>\fontspec_if_script:nTF</code>	Test whether the currently selected font contains the raw OpenType script #1. E.g.: <code>\fontspec_if_script:nTF {latn} {True} {False}</code> . Returns false if the font is not loaded by fontspec or is not an OpenType font.
<code>\fontspec_if_language:nTF</code>	Test whether the currently selected font contains the raw OpenType language tag #1. E.g.: <code>\fontspec_if_language:nTF {ROM} {True} {False}</code> . Returns false if the font is not loaded by fontspec or is not an OpenType font.
<code>\fontspec_if_language:nnTF</code>	Test whether the currently selected font contains the raw OpenType language tag #2 in script #1. E.g.: <code>\fontspec_if_language:nnTF {cyr1} {SRB} {True} {False}</code> . Returns false if the font is not loaded by fontspec or is not an OpenType font.

19.3.3 Currently selected features

<code>\fontspec_if_current_feature:nTF</code>	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.
<code>\fontspec_if_current_script:nTF</code>	Test whether the currently loaded font is using the specified raw OpenType script tag #1.
<code>\fontspec_if_current_language:nTF</code>	Test whether the currently loaded font is using the specified raw OpenType language tag #1.

Part VIII

The ‘improvement’ of L^AT_EX 2_ε and other packages

This part of the package code contains patches to various L^AT_EX components and third-party packages to improve the default behaviour.

20 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, L^AT_EX no longer prints visible spaces for the `verbatim*` environment and `\verb*` command. This problem is fixed by using the correct Unicode glyph, and the following packages are patched to do the same: `listings`, `fancyvrb`, `moreverb`, and `verbatim`.

In the case that the typewriter font does not contain ‘`□`’, the Latin Modern Mono font is used as a fallback.

21 Discretionary hyphenation: `\-`

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

22 Commands for old-style and lining numbers

`\oldstylenums` L^AT_EX’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>}`.