The fontspec package
Font selection for \texttt{XeLaTeX} and \texttt{LuaLaTeX}

WILL ROBERTSON
With contributions by Khaled Hosny, Philipp Gesang, Joseph Wright, and others.

http://wspr.io/fontspec/

2020/02/21 v2.7i

Contents

I Getting started 5
1 History 5

2 Introduction 5
  2.1 Acknowledgements ........................................ 5

3 Package loading and options 6
  3.1 Font encodings ............................................. 6
  3.2 Maths fonts adjustments .................................... 6
  3.3 Configuration ............................................... 6
  3.4 Warnings .................................................... 6

4 Interaction with \LaTeX\ and other packages 7
  4.1 Commands for old-style and lining numbers .............. 7
  4.2 Italic small caps .......................................... 7
  4.3 Emphasis and nested emphasis .............................. 7
  4.4 Strong emphasis ........................................... 7

II General font selection 8

1 Main commands 8

2 Font selection 9
  2.1 By font name ............................................... 9
  2.2 By file name ................................................ 10
  2.3 By custom file name using a \texttt{.fontspec} file ....... 11
  2.4 Querying whether a font `exists' ........................ 12
3 Commands to select font families 13
4 Commands to select single font faces 13
   4.1 More control over font shape selection 14
   4.2 Specifically choosing the \texttt{nss} family 15
   4.3 Choosing additional \texttt{nss} font faces 16
   4.4 Math(s) fonts 17
5 Miscellaneous font selecting details 18

III Selecting font features 19
1 Default settings 19
2 Working with the currently selected features 20
   2.1 Priority of feature selection 21
3 Different features for different font shapes 21
4 Selecting fonts from TrueType Collections (TTC files) 23
5 Different features for different font sizes 23
6 Font independent options 24
   6.1 Colour 25
   6.2 Scale 25
   6.3 Interword space 26
   6.4 Post-punctuation space 27
   6.5 The hyphenation character 27
   6.6 Optical font sizes 28
   6.7 Font transformations 28
   6.8 Letter spacing 30

IV OpenType 31
1 Introduction 31
   1.1 How to select font features 31
   1.2 How do I know what font features are supported by my fonts? 32
2 OpenType scripts and languages 33
   2.1 Script and Language examples 33
3 OpenType font features 33
   3.1 Tag-based features 36
   3.2 CJK features 45
V Commands for accents and symbols ('encodings')
1 A new Unicode-based encoding from scratch
2 Adjusting a pre-existing encoding
3 Summary of commands

VI Lua\TeX-only font features
1 Different font technologies and shapers
2 Custom font features

VII Fonts and features with X\TeX
1 X\TeX-only font features
1.1 Mapping
1.2 Different font technologies: AAT, OpenType, and Graphite
1.3 Optical font sizes
1.4 Vertical typesetting
2 The Graphite renderer
3 macOS's AAT fonts
3.1 Ligatures
3.2 Letters
3.3 Numbers
3.4 Contextuals
3.5 Vertical position
3.6 Fractions
3.7 Variants
3.8 Alternates
3.9 Style
3.10 CJK shape
3.11 Character width
3.12 Diacritics
3.13 Annotation

VIII Customisation and programming interface
1 Defining new features
2 Defining new scripts and languages
3 Going behind fontspec's back
4 Renaming existing features & options  63

5 Programming interface  63
  5.1 Variables ................................................. 63
  5.2 Functions for loading new fonts and families .......... 63
  5.3 Conditionals .............................................. 64
Part I
Getting started

1 History

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

More recently, 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 Con\TeX{}Xt Mk. IV is a re-write of his powerful typesetting system, taking full advantage of 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 Lua\TeX{} engine.

2 Introduction

The fontspec package allows users of either \TeX{}X or 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 \texttt{\emph{}} or \texttt{\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 \TeX{}X should have little or no difficulty switching over to 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 \TeX{}X, 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.

3.1 Font encodings

The (default) tuenc package option switches the NFSS font encoding to TU. TU is a new Unicode font encoding, intended for both \texttt{Xe\LaTeX} and \texttt{Lua\LaTeX} engines, and automatically contains support for symbols covered by \LaTeX's traditional \texttt{T1} and \texttt{TS1} font encodings (for example, \texttt{\textbackslash%}, \texttt{\textbullet}, \texttt{"u}, and so on). Some additional features are provided by fontspec to customise some encoding details; see Part V on page 50 for further details.

Pre-2017 behaviour can be achieved with the euenc package option. This selects the \texttt{EU1} or \texttt{EU2} encoding (\texttt{Xe\LaTeX}/\texttt{Lua\LaTeX}, resp.) and loads the \texttt{xunicode} package for symbol support. Package authors and users who have referred explicitly to the encoding names \texttt{EU1} or \texttt{EU2} should update their code or documents. (See internal variable names described in Section 5 on page 63 for how to do this properly.)

3.2 Maths fonts adjustments

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

If you find that fontspec is incorrectly changing the maths font when it shouldn’t be, apply the no-math package option to manually suppress its behaviour here.

3.3 Configuration

If you wish to customise any part of the fontspec interface, this should be done by creating your own \texttt{fontspec.cfg} file, which will be automatically loaded if it is found by \texttt{Xe\LaTeX} or \texttt{Lua\LaTeX}. A \texttt{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 \texttt{.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 \texttt{Xe\LaTeX} or \texttt{Lua\LaTeX} searches by default; \texttt{e.g}. in \texttt{Mac\LaTeX}: \texttt{~/Library/texmf/tex/latex/}.

The package option \texttt{no-config} will suppress the loading of the \texttt{fontspec.cfg} file under all circumstances.

3.4 Warnings

This package can give some warnings that can be harmless if you know what you’re doing. Use the quiet package option to write these warnings to the transcript (\texttt{.log}) file instead.

Use the silent package option to completely suppress these warnings if you don’t even want the \texttt{.log} file cluttered up.
4 Interaction with \LaTeX\ 2\epsilon and other packages

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

4.1 Commands for old-style and lining numbers

\texttt{\LaTeX}’s definition of \texttt{\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 \texttt{\oldstylenums{(text)}} to explicitly use old-style (or lowercase) numbers in \texttt{(text)}, and the reverse for \texttt{\liningnums{(text)}}.

4.2 Italic small caps

Support now provided by \LaTeX\ 2\epsilon in 2020.

4.3 Emphasis and nested emphasis

Support now provided by \LaTeX\ 2\epsilon in 2020.

4.4 Strong emphasis

The \texttt{\strong} macro is used analogously to \texttt{\emph} but produces variations in weight. If you need it in environment form, use \texttt{\begin{strongenv}{\ldots} \end{strongenv}}.

As with emphasis, this font-switching command is intended to move through a range of font weights. For example, if the fonts are set up correctly it allows usage such as \texttt{\strong{\ldots}\strong{\ldots}} in which each nested \texttt{\strong} macro increases the weight of the font.

Currently this feature set is somewhat experimental and there is no syntactic sugar to easily define a range of font weights using fontspec commands. Use, say, the following to define first bold and then black (\texttt{k}) font faces for \texttt{\strong}:

\begin{verbatim}
\strongfontdeclare{\bfseries,\fontseries(k)\selectfont}
\end{verbatim}

If too many levels of \texttt{\strong} are reached, \texttt{\strongreset} is inserted. By default this is a no-op and the font will simply remain the same. Use \texttt{\renewcommand{\strongreset}{\mdseries}} to start again from the beginning if desired.

An example for setting up a font family for use with \texttt{\strong} is discussed in 4.3.1 on page 17.
Part II

General font selection

1 Main commands

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

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

These are the main font-selecting commands of this package which select the standard fonts used in a document, as shown in Example 1. Here, the scales of the fonts have been chosen to equalise their lowercase letter heights. The Scale font feature will be discussed further in Section 6 on page 24, including methods for automatic scaling. Note that further options may need to be added to select appropriate bold/italic fonts, but this shows the main idea.

Note that while these commands all look and behave largely identically, the default setup for font loading automatically adds the Ligatures=TeX feature for the \setmainfont and \setsansfont commands. These defaults (and further customisations possible) are discussed in Section 1 on page 19.

\newfontfamily⟨cmd⟩{⟨font⟩}[⟨font features⟩]
\setfontfamily⟨cmd⟩{⟨font⟩}[⟨font features⟩]
\renewfontfamily⟨cmd⟩{⟨font⟩}[⟨font features⟩]
\providefontfamily⟨cmd⟩{⟨font⟩}[⟨font features⟩]

These commands define new font family commands (like \rmfamily). The new command checks if ⟨cmd⟩ has been defined, and issues an error if so. The renew command checks if ⟨cmd⟩ has been defined, and issues an error if not. The provide command checks if ⟨cmd⟩ has been defined, and silently aborts if so. The set command never checks; use at your own risk.

\fontspec{⟨font⟩}[⟨font features⟩]

The plain \fontspec command is not generally recommended for document use. It is an ad hoc command best suited for testing and loading fonts on a one-off basis.

All of the commands listed above accept comma-separated ⟨font feature⟩=⟨option⟩ lists; these are described later:

- For general font features, see Section 6 on page 24
- For OpenType fonts, see Part IV on page 31
- For XeLaTeX-only general font features, see Part VII on page 56
- For LuaLaTeX-only general font features, see Part VI on page 54
- For features for AAT fonts in XeLaTeX, see Section 3 on page 58
Example 1: Loading the default, sans serif, and monospaced fonts.

\setmainfont{texgyrebonum-regular.otf}
\setsansfont{lmsans18-regular.otf}[Scale=MatchLowercase]
\setmonofont{Inconsolatazi4-Regular.otf}[Scale=MatchLowercase]
\rmfamily Pack my box with five dozen liquor jugs\par
\sffamily Pack my box with five dozen liquor jugs\par
\ttfamily Pack my box with five dozen liquor jugs

2 Font selection

In both \LaTeX{} and \XeLaTeX{}, fonts can be selected (using the \textit{font} argument in Section 1) 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.

2.1 By font name

Fonts known to \LaTeX{} or \XeLaTeX{} may be loaded by their standard names as you’d speak them out loud, such as \textit{Times New Roman} or \textit{Adobe Garamond}. ‘Known to’ in this case generally means ‘exists in a standard fonts location’ such as ~/Library/Fonts on macOS, or C:\Windows\Fonts on Windows. In \LaTeX{}, fonts found in the \texttt{TEXMF} tree can also be loaded by name. In \XeLaTeX{}, fonts found in the \texttt{TEXMF} tree can be loaded in Windows and Linux, but not on macOS.

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 \texttt{\textit{}} and \texttt{\textbf{}} commands.

The ‘font name’ can be found in various ways, such as by looking in the name listed in an application like \texttt{Font Book} on Mac OS X. Alternatively, \TeX{}Live contains the \texttt{otfinfo} command line program, which can query this information; for example:

\texttt{otfinfo -i `kpsewhich lmroman10-regular.otf`}

results in a line that reads:

\begin{verbatim}
Preferred family: Latin Modern Roman
\end{verbatim}

(The ‘preferred family' name is usually better than the ‘family' name.)

\LaTeX{} users only In order to load fonts by their name rather than by their filename (e.g., ‘Latin Modern Roman’ instead of ‘ec-lmr10’), you may need to run the script \texttt{luaotfload-tool}, which is distributed with the \texttt{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 \texttt{luaotfload} documentation for more information.
2.2 By file name

Xe\TeX{} and \LaTeX{} 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 Xe\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 Xe\TeX{} but cannot be loaded by font name, only file name; \LaTeX{} 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 file name must include bold and italic variants explicitly, unless a .\fontspec file is supplied for the font family (see Section 2.3). We’ll give some first examples specifying everything explicitly:

\begin{verbatim}
\setmainfont{texgyrepagella-regular.otf}[
  BoldFont = texgyrepagella-bold.otf ,
  ItalicFont = texgyrepagella-italic.otf ,
  BoldItalicFont = texgyrepagella-bolditalic.otf ]
\end{verbatim}

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

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

If desired, an abbreviation can be applied to the font names based on the mandatory \texttt{font name} argument:

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

In this case \texttt{texgyrepagella} is no longer the name of an actual font, but is used to construct the font names for each shape; the \texttt{*} is replaced by \texttt{texgyrepagella}. Note in this case that \texttt{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 \texttt{Path} feature:

\begin{verbatim}
\setmainfont{texgyrepagella}[
  Path = /Users/will/Fonts/ ,
  UprightFont = *-regular ,
  BoldFont = *-bold ,
  ... ]
\end{verbatim}
Note that XeLaTeX and LuaLaTeX 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}\[
  \text{Path, \textit{BoldFont = texgyrepagella-bold,} ...}
\]

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.

2.3 By custom file name using a \texttt{.fontspec} file

When fontspec is first asked to load a font, a font settings file is searched for with the name ‘⟨\textit{fontname}.fontspec⟩’\footnote{Located in the current folder or within a standard \texttt{texmf} location.}. If you want to disable this feature on a per-font basis, use the IgnoreFontspecFile font option.

The contents of this file can be used to specify font shapes and font features without having to have this information present within each document. Therefore, it can be more flexible than the alternatives listed above.

When searching for this \texttt{.fontspec} file, ⟨\textit{fontname}⟩ is stripped of spaces and file extensions are omitted. For example, given \texttt{\setmainfont{TeX Gyre Adventor}}, the \texttt{.fontspec} file would be called \texttt{TeXGyreAdventor.fontspec}. If you wanted to transparently load options for \texttt{\setmainfont{texgyreadventor-regular.otf}}, the configuration file would be \texttt{texgyreadventor-regular.fontspec}.

N.B. that while spaces are stripped, the lettercase of the names should match.

This mechanism can be used to define custom names or aliases for your font collections. The syntax within this file follows from the \texttt{\defaultfontfeatures}, defined in more detail later but mirroring the standard fontspec font loading syntax. As an example, suppose we’re defining a font family to be loaded with \texttt{\setmainfont{My Charis}}. The corresponding \texttt{MyCharis.fontspec} file would contain, say,

\texttt{\defaultfontfeatures[My Charis]}

\{  
  \text{Extension = .ttf ,}  
  \text{UprightFont = CharisSILR,}  
  \text{BoldFont = CharisSILB,}  
  \text{ItalicFont = CharisSILI,}  
  \text{BoldItalicFont = CharisSILBI,}  
  \% <any other desired options>
\}

The optional argument to \texttt{\defaultfontfeatures} must exactly match that requested by the font loading command (\texttt{\setmainfont}, etc.) — in particular note that spaces are significant here, so \texttt{\setmainfont{MyCharis}} will not ‘see’ the default font feature setting within the \texttt{.fontspec} file.
Finally, note that options for individual font faces can also be defined in this way. To continue the example above, here we colour the different faces:

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

Such configuration lines could be stored either inline inside My Charis.fontspec or within their own .fontspec files; in this way, fontspec is designed to handle ‘nested’ configuration options.

Where \defaultfontfeatures is being used to specify font faces by a custom name, the Font feature is used to set the filename of the font face. For example:

\defaultfontfeatures[charis]
{
  UprightFont = charis-regular,
  % <other desired options for all font faces in the family>
}
\defaultfontfeatures[charis-regular]
{
  Font = CharisSILR
  % <other desired options just for the `upright' font>
}

The fontspec interface here is designed to be flexible to accommodate a variety of use cases; there is more than one way to achieve the same outcome when font faces are collected together into a larger font family.

2.4 Querying whether a font ‘exists’

\IfFontExistsTF\{⟨font name⟩\}{\langle true branch\rangle}{\langle false branch\rangle}

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

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

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

E.g.:

\IfFontExistsTF{cmr10}{T}{F}
\IfFontExistsTF{times new roman}{T}{F}
\IfFontExistsTF{texgyrepagella-regular.otf}{T}{F}
\IfFontExistsTF{/Users/will/Library/Fonts/CODE2000.TTF}{T}{F}

The \IfFontExistsTF command is a synonym for the programming interface function \fontspec_font_if_exist:nTF (Section 5 on page 63).
3 Commands to select font families

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 \texttt{\fontspec} for every use. While the \texttt{\fontspec} command does not define a new font instance after the first call, the feature options must still be parsed and processed.

For this reason, new commands can be created for loading a particular font family with the \texttt{\newfontfamily} command and variants, outlined in Section 1 on page 8 and demonstrated in Example 2. This macro should be used to create commands that would be used in the same way as \texttt{\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 \texttt{\emph}) define it using regular \LaTeX{} commands:

\begin{verbatim}
\newcommand\textnote[1]{{\notefont #1}}
\textnote{This is a note.}
\end{verbatim}

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

Comment for advanced users: The commands defined by \texttt{\newfontfamily} (and \texttt{\newfontface}; see next section) 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,

\begin{verbatim}
\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}
\end{verbatim}

4 Commands to select single font faces

\begin{verbatim}
\newfontface\texttt{cmd}{(font)}{(font features)}
\setfontface\texttt{cmd}{(font)}{(font features)}
\renewfontface\texttt{cmd}{(font)}{(font features)}
\providefontface\texttt{cmd}{(font)}{(font features)}
\end{verbatim}

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. \texttt{\newfontface} is used for this purpose, shown in Example 3, which is repeated in Section 3.4 on page 59.

\begin{verbatim}
\newfontfamily\notefont{Kurier}
\notefont This is a \emph{note}.
\end{verbatim}
Example 3: Defining a single font face.

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

4.1 More control over font shape selection

| \texttt{BoldFont} = \langle \texttt{font name} \rangle |
| \texttt{ItalicFont} = \langle \texttt{font name} \rangle |
| \texttt{BoldItalicFont} = \langle \texttt{font name} \rangle |
| \texttt{SlantedFont} = \langle \texttt{font name} \rangle |
| \texttt{BoldSlantedFont} = \langle \texttt{font name} \rangle |
| \texttt{SmallCapsFont} = \langle \texttt{font name} \rangle |
| \texttt{UprightFont} = \langle \texttt{font name} \rangle |

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 \texttt{BoldFont} and \texttt{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 \texttt{BoldItalicFont} feature is provided.

4.1.1 Small caps and slanted font shapes

When a font family has both slanted \texttt{and} italic shapes, these may be specified separately using the analogous features \texttt{SlantedFont} and \texttt{BoldSlantedFont}. Without these, however, the \LaTeX{} font switches for slanted (\texttt{textsl}, \texttt{slshape}) will default to the italic shape.

For modern OpenType fonts, small caps glyphs are included within a fontface and \texttt{fontspec} will automatically detect them for use with the \texttt{textsc} and \texttt{scshape} commands. 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 \texttt{SmallCapsFont} of the family you are specifying:

Example 4: Explicit selection of the bold font.

<table>
<thead>
<tr>
<th>Helvetica Neue UltraLight</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{Helvetica Neue UltraLight}</td>
</tr>
<tr>
<td>\texttt{Helvetica Neue UltraLight Italic}</td>
</tr>
<tr>
<td>Helvetica Neue</td>
</tr>
<tr>
<td>Helvetica Neue Italic</td>
</tr>
</tbody>
</table>

\texttt{\fontspec{Helvetica Neue UltraLight}%
[\texttt{BoldFont=\{Helvetica Neue\}}]
\texttt{Helvetica Neue UltraLight}} \texttt{\%}
\texttt{Helvetica Neue UltraLight Italic}} \texttt{\%}
\texttt{\{\texttt{bfseries}} \texttt{Helvetica Neue}} \texttt{\%}
\texttt{\{\texttt{bfseries\textitshape}} \texttt{Helvetica Neue Italic}}} \texttt{\%}
In fact, this example is overly simplistic since it does not cover the other shapes in a font family. 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> } } ,  
]

For most modern fonts that have small caps as a font feature, this level of control isn’t generally necessary.

All of the bold, italic, and small caps fonts can be loaded with different font features from the main font. See Section 3 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},  
]

4.2 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 fontspe, 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 you 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 fontspe use the NFSSFamily feature:

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

\footnote{Thanks to Luca Fascione for the example and motivation for finally implementing this feature.}
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 \texttt{N}SSS 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 \texttt{\fontspec_set_family:Nnn} for details (Section \texttt{5} on page \texttt{63}).

4.3 Choosing additional \texttt{N}SSS font faces

\LaTeX{}\texttt{X}’s font selection scheme (\texttt{N}SSS) is more flexible than the fontspec interface discussed up until this point. It assigns to each font face a \texttt{family} (discussed above), a \texttt{series} such as bold or light or condensed, and a \texttt{shape} such as italic or slanted or small caps. The fontspec features such as \texttt{BoldFont} and so on all assign faces for the default series and shapes of the \texttt{N}SSS, 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 \texttt{\setmainfont{font1.otf}}[\texttt{\FontFace = \{c\}\{\shapedefault\}\{\font2.otf\},\ FontFace = \{c\}\{m\}\{\font3.otf,\ Color = \textcolor{red}{red}\}}]

Writing \texttt{\fontseries{c}\selectfont} will result in \texttt{font2} being selected, which then followed by \texttt{\fontshape{m}\selectfont} will result in \texttt{font3} being selected (in red). A font face that is defined in terms of a different series but an upright shape (\texttt{\shapedefault}, as shown above) will attempt to find a matching small caps feature and define that face as well. Conversely, a font face defined in terms of a non-standard font shape will not.

There are some standards for choosing shape and series codes; the \LaTeX{}\texttt{2e} font selection guide\textsuperscript{3} has a comprehensive listing.

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

\texttt{\setmainfont{font1.otf}[
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 \texttt{(features)} as requested. (Note that the optional \texttt{\{\{features\}\}} argument is still surrounded with curly braces.) Multiple \texttt{FontFace} commands may be used in a single declaration to specify multiple fonts. As an example:

\texttt{\setmainfont{font1.otf}[
FontFace = \{c\}\{\shapedefault\}\{ font2.otf \},
FontFace = \{c\}\{m\}\{ Font = font3.otf , Color = \textcolor{red}{red} \}
]}

Writing \texttt{\fontseries{c}\selectfont} will result in \texttt{font2} being selected, which then followed by \texttt{\fontshape{m}\selectfont} will result in \texttt{font3} being selected (in red). A font face that is defined in terms of a different series but an upright shape (\texttt{\shapedefault}, as shown above) will attempt to find a matching small caps feature and define that face as well. Conversely, a font face defined in terms of a non-standard font shape will not.

There are some standards for choosing shape and series codes; the \LaTeX{}\texttt{2e} font selection guide\textsuperscript{3} has a comprehensive listing.

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

\textsuperscript{3}\texttt{texdoc fntguide}
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.

4.3.1 An example for \strong

If you wanted to set up a font family to allow nesting of the \strong to easily access increasing font weights, you might use a declaration along the following lines:

\setmonofont{SourceCodePro}\[
  Extension = .otf ,
  UprightFont = **-Light ,
  BoldFont = **-Regular ,
  FontFace = \{k\}{n}\{**-Black\} ,
\]
\strongfontdeclare{\bfseries,\fontseries{k}\selectfont}

Further ‘syntactic sugar’ is planned to make this process somewhat easier.

4.4 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.)

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

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

\setmathrm{⟨font name⟩}\[⟨font features⟩]\n\setmathsf{⟨font name⟩}\[⟨font features⟩]\n\setmathtt{⟨font name⟩}\[⟨font features⟩]\n\setboldmathrm{⟨font name⟩}\[⟨font features⟩]\n
4 Speaking of euler, if you want to use its [mathbf] option, it won’t work, and you’ll need to put this after fontspec is loaded instead: \AtBeginDocument{\DeclareMathAlphabet\mathbf{U}{eur}{b}{n}}
However, the default text fonts may not necessarily be the ones you wish to use when typesetting maths (especially with the use of fancy ligatures and so on). For this reason, you may optionally use the commands above (in the same way as our other \fontspec-like commands) to explicitly state which fonts to use inside such commands as \mathrm. Additionally, the \setboldmathrm command allows you define the font used for \mathrm when in bold maths mode (which is activated with, among others, \boldmath).

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

\begin{verbatim}
\usepackage{mathpazo}
\usepackage{fontspec}
\setmainfont{Optima}
\setmathrm{Optima}
\setboldmathrm{BoldFont={Optima ExtraBlack}}{Optima Bold}
\end{verbatim}

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.

5 Miscellaneous font selecting details

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

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

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:

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

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

\begin{verbatim}
\fontspec[Ligatures=Rare]{myfont.otf}[Color=red]
\end{verbatim}

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

Spaces \fontspec and \addfontfeatures ignore trailing spaces as if it were a ‘naked’ control sequence; e.g., M. \fontspec{...} W and M. \fontspec{...}W are the same.
Part III
Selecting font features

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

1 Default settings

\defaultfontfeatures{⟨font features⟩}

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

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

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

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

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

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

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

\defaultfontfeatures[\rmfamily, \sffamily]{Ligatures=TeX}
\setmainfont{texgxyreadventor-regular.otf}% will use standard TeX ligatures

Example 5: A demonstration of the \defaultfontfeatures command.

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

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

19
The line above to set TeX-like ligatures is now activated by default in fontspec.cfg. To reset default font features, simply call the command with an empty argument:

\defaultfontfeatures[\rmfamily,\sffamily]{}
\setmainfont{texgyreadventor-regular.otf}% will no longer use standard TeX ligatures

New in v2.4. Using the + form of the command appends the \textit{font features} to any already-selected defaults.

2 Working with the currently selected features

\IfFontFeatureActiveTF{\textit{font feature}}{\textit{true code}}{\textit{false code}}

This command queries the currently selected font face and executes the appropriate branch based on whether the \textit{font feature} as specified by fontspec is currently active.

For example, the following will print 'True':

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

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

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

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

\addfontfeatures{\textit{font features}}

This command allows font features in an entire font family to be changed without knowing what features are currently selected or even what font family is being used. A good example of this could be to add a hook to all tabular material to use monospaced numbers, as shown in Example 6. If you attempt to change an already-selected feature, fontspec will try to de-activate any features that clash with the new ones. E.g., the following two invocations are mutually exclusive:

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

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

If you wish to apply the change to only one of the fonts of a family (say, italics only) you can write

\addfontfeature{ItalicFeatures={Numbers=Lowercase}}

This command may also be executed under the alias \addfontfeature.
Example 6: A demonstration of the `\addfontfeatures` command.

```
\fontspec{texgyreadventor-regular.otf}\
[Numbers={Proportional,OldStyle}]
'In 1842, 999 people sailed 97 miles in 13 boats. In 1923, 111 people sailed 54 miles in 56 boats.'
\bigskip
\addfontfeatures{Numbers={Monospaced,Lining}}
\begin{tabular}{@{} cccc @{}}
  Year & People & Miles & Boats \\
 1842 & 999 & 75 & 13 \\
 1923 & 111 & 54 & 56 \\
\end{tabular}
```

2.1 Priority of feature selection

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

3 Different features for different font shapes

<table>
<thead>
<tr>
<th>BoldFeatures</th>
<th>{⟨features⟩}</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItalicFeatures</td>
<td>{⟨features⟩}</td>
</tr>
<tr>
<td>BoldItalicFeatures</td>
<td>{⟨features⟩}</td>
</tr>
<tr>
<td>SlantedFeatures</td>
<td>{⟨features⟩}</td>
</tr>
<tr>
<td>BoldSlantedFeatures</td>
<td>{⟨features⟩}</td>
</tr>
<tr>
<td>SmallCapsFeatures</td>
<td>{⟨features⟩}</td>
</tr>
<tr>
<td>UprightFeatures</td>
<td>{⟨features⟩}</td>
</tr>
</tbody>
</table>

It is entirely possible that separate fonts in a family will require separate options; e.g., Hoefler Text Italic contains various swash feature options that are completely unavailable in the upright shapes.

The font features defined at the top level of the optional `\fontspec` argument are applied to all shapes of the family. Using `Upright-`, `SmallCaps-`, `Bold-`, `Italic-`, and `BoldItalicFeatures`, separate font features may be defined to their respective shapes in addition to, and with precedence over, the `global` font features. See Example 7.

Note that because most fonts include their small caps glyphs within the main font, features specified with `SmallCapsFeatures` are applied in addition to any other shape-specific features as defined above, and hence `SmallCapsFeatures` can be nested within `ItalicFeatures` and friends. Every combination of upright, italic, bold and small caps can thus be assigned individual features, as shown in the somewhat ludicrous Example 8.
Example 7: Features for, say, just italics.

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

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

\fontspec{texgyretermes}[
  Extension = {.otf},
  UprightFont = {*-regular}, ItalicFont = {*-italic},
  BoldFont = {*-bold}, BoldItalicFont = {*-bolditalic},
  UprightFeatures={Color = 220022,
                  SmallCapsFeatures = {Color=115511}},
  ItalicFeatures={Color = 2244FF, 
                 SmallCapsFeatures = {Color=112299}},
  BoldFeatures={Color = FF4422, 
               SmallCapsFeatures = {Color=992211}},
  BoldItalicFeatures={Color = 888844, 
                  SmallCapsFeatures = {Color=444422}},
  ]

Upright \textsc{Small Caps}
Italic \textit{Italic Small Caps}
Bold \textbf{Bold Small Caps}
Bold Italic \textit{Bold Italic Small Caps}
Selecting fonts from TrueType Collections (TTC files)

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

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

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

Different features for different font sizes

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

It takes a comma separated list of braced, comma separated lists of features for each size range. Each sub-list must contain the `Size` option to declare the size range, and optionally `Font` to change the font based on size. Other (regular) fontspec features that are added are used on top of the font features that would be used anyway. A demonstration to clarify these details is shown in Example 9. A less trivial example is shown in the context of optical font sizes in Section 6.6 on page 28.

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

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

```latex
SizeFeatures={
    {Size=-10,Numbers=Uppercase},
    {Size=10-}}
```
Example 9: An example of specifying different font features for different sizes of font with SizeFeatures.

```
\fontspec{texgyrechorus-mediumitalic.otf}{
  SizeFeatures={
    \{Size={-8}, Font=texgyrebonum-italic.otf, Color=AA0000},
    \{Size={8-14}, Color=00AA00},
    \{Size={14-}, Color=0000AA}\}
}\scriptsize Small\par Normal size\par \Large Large\par
```

Otherwise, the font sizes greater than 10 won’t be defined at all!

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

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

Suggestions on simplifying this interface welcome.

### 6 Font independent options

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

<table>
<thead>
<tr>
<th>Input</th>
<th>Font size, s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size = X-</td>
<td>$s \geq X$</td>
</tr>
<tr>
<td>Size = -Y</td>
<td>$s &lt; Y$</td>
</tr>
<tr>
<td>Size = X-Y</td>
<td>$X \leq s &lt; Y$</td>
</tr>
<tr>
<td>Size = X</td>
<td>$s = X$</td>
</tr>
</tbody>
</table>
6.1 Colour

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

The colour is defined as a triplet of two-digit Hex RGB values, with optionally another value for the transparency (where 00 is completely transparent and FF is opaque.) Transparency is supported by LuaLaTeX; XeLaTeX with the xdvipdfmx driver does not support this feature.

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

\usepackage{xcolor}

\begin{verbatim}
\fontspec[Color=red]{Verdana} ...
\definecolor{Foo}{rgb}{0.3,0.4,0.5}
\fontspec[Color=Foo]{Verdana} ...
\end{verbatim}

The color package is not supported; use xcolor instead.

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

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

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

6.2 Scale

\begin{verbatim}
\begin{tabular}{l}
Scale = (number) \\
Scale = MatchLowercase \\
Scale = MatchUppercase
\end{tabular}
\end{verbatim}

In its explicit form, Scale takes a single numeric argument for linearly scaling the font, as demonstrated in Example 1.

---

Example 10: Selecting colour with transparency.

\begin{verbatim}
\fontsize{48}{48}\selectfont
\fontspec{texgyrebonum-bold.otf}
{\addfontfeature{Color=FF000099}\kern=-0.4ex}
{\addfontfeature{Color=0000FF99}\kern=-0.4ex}
{\addfontfeature{Color=DDBB2299}\kern=-0.5ex}
{\addfontfeature{Color=00BB3399}\kern=-0.5ex}
\end{verbatim}
As well as a numerical argument, the Scale feature also accepts options MatchLowercase and MatchUppercase, which will scale the font being selected to match the current default roman font to either the height of the lowercase or uppercase letters, respectively; these features are shown in Example 11. The amount of scaling used in each instance is reported in the .log file.

Additional calls to the Scale feature overwrite the settings of the former. If you want to accumulate scale factors (useful perhaps to fine-tune the settings of MatchLowercase), the ScaleAgain feature can be used as many times as necessary. For example:

\[ \text{Scale} = 1.1, \text{Scale} = 1.2 \] \% \rightarrow \text{scale of 1.2}
\[ \text{Scale} = 1.1, \text{ScaleAgain} = 1.2 \] \% \rightarrow \text{scale of 1.32}

Note that when Scale=MatchLowercase is used with \setmainfont, the new ‘main’ font of the document will be scaled to match the old default. If you wish to automatically scale all fonts except have the main font use ‘natural’ scaling, you may write

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

One or both of these lines may be placed into a local fontspec.cfg file (see Section 3.3 on page 6) for this behaviour to be effected in your own documents automatically. (Also see Section 1 on page 19 for more information on setting font defaults.)

6.3 Interword space

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

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

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

---

<table>
<thead>
<tr>
<th>Example 11: Automatically calculated scale values.</th>
</tr>
</thead>
</table>
| The perfect match is hard to find. \begin{verbatim}
   LOGOFONT
   \setmainfont{Georgia}
   \newfontfamily[lc][Scale=MatchLowercase]{Verdana}
   The perfect match \{lc is hard to find.}\n\end{verbatim}
| \newfontfamily[uc][Scale=MatchUppercase]{Arial}
   LOGO \uc F O N T |
| 26 |
Example 12: Scaling the default interword space. An exaggerated value has been chosen to emphasise the effects here.

\fontspec{texgyretermes-regular.otf}
Some text for our example to take up some space, and to demonstrate the default interword space.
\bigskip
\fontspec{texgyretermes-regular.otf}\[WordSpace = 0.3\]
Some text for our example to take up some space, and to demonstrate the default interword space.

6.4 Post-punctuation space

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

The \texttt{PunctuationSpace} feature takes a scaling factor by which to adjust the nominal value chosen for the font; this is demonstrated in Example 13. Note that \texttt{PunctuationSpace=0} is not equivalent to \texttt{\frenchspacing}, although the difference will only be apparent when a line of text is under-full.

Note that \LaTeX{}'s optimisations in how it loads fonts means that you cannot use this feature in \texttt{\addfontfeatures}.

6.5 The hyphenation character

The letter used for hyphenation may be chosen with the \texttt{HyphenChar} feature. With one exception (\texttt{HyphenChar = None}), this is a \LaTeX{}-only feature since \texttt{Lua\TeX{}} cannot set the hyphenation character on a per-font basis; see its \texttt{\prehyphenchar} primitive for further details.

\texttt{HyphenChar} takes three types of input, which are chosen according to some simple rules. If the input is the string \texttt{None}, then hyphenation is suppressed for this font.

Example 13: Scaling the default post-punctuation space.

\nonfrenchspacing
\fontspec{texgyreschola-regular.otf}Letters, Words. Sentences. \par
\fontspec{texgyreschola-regular.otf}[PunctuationSpace=2]Letters, Words. Sentences. \par
\fontspec{texgyreschola-regular.otf}[PunctuationSpace=0]Letters, Words. Sentences.
As part of fontspec.cfg, the default monospaced family (e.g., \ttfamily) is set up to automatically set \texttt{HyphenChar = None}.

If the input is a single character, then this character is used. Finally, if the input is longer than a single character it must be the UTF-8 slot number of the hyphen character you desire.

Note that \TeX{}'s optimisations in how it loads fonts means that you cannot use this feature in \texttt{\addfontfeatures}.

### 6.6 Optical font sizes

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

OpenType fonts with optical scaling can exist in several discrete sizes (in separate font files). When loading fonts by name, X\TeX{} and Lua\TeX{} engines will attempt to automatically load the appropriate font as determined by the current font size. An example of this behaviour is shown in Example 15, in which some larger text is mechanically scaled down to compare the difference for equivalent font sizes.

The \texttt{OpticalSize} feature may be used to specify a different optical size. With \texttt{OpticalSize} set (Example 16) to zero, no optical size font substitution is performed.

The \texttt{SizeFeatures} feature (Section 5 on page 23) can be used to specify exactly which optical sizes will be used for ranges of font size. For example, something like:

\begin{verbatim}
\fontspec{Latin Modern Roman}\{
  UprightFeatures = \{ SizeFeatures = \{
    \{Size=-10, OpticalSize=8\},
    \{Size=10-14, OpticalSize=10\},
    \{Size=14-18, OpticalSize=14\},
    \{Size=18-, OpticalSize=18\}\}\}
\end{verbatim}

### 6.7 Font transformations

In rare situations users may want to mechanically distort the shapes of the glyphs in the current font such as shown in Example 17. Please don’t overuse these features; they are not a good alternative to having the real shapes.

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

---

**Example 14:** Explicitly choosing the hyphenation character.

```latex
\text{\fbox{\parbox{1.55cm}{%}
  \texttt{\% EX\text{\small E}MPL\text{\small E}LE HYPHENATION}\%}}\quad\quad\null\par\bigskip
\text{\fontspec{LinLibertine_R.otf}[HyphenChar=None]}
\text{\fontspec{LinLibertine_R.otf}[HyphenChar=+]}
```

28
### Example 15: A demonstration of automatic optical size selection.

<table>
<thead>
<tr>
<th>Automatic optical size</th>
<th>\texttt{\fontspec{Latin Modern Roman}} \texttt{Automatic optical size}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>\texttt{\scalebox{0.4}\Huge Automatic optical size}</td>
</tr>
</tbody>
</table>

### Example 16: Explicit optical size substitution for the Latin Modern Roman family.

<table>
<thead>
<tr>
<th>Latin Modern optical sizes</th>
<th>\texttt{\fontspec{Latin Modern Roman}[OpticalSize=5]}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>\texttt{Latin Modern optical sizes}</td>
</tr>
<tr>
<td></td>
<td>\texttt{\fontspec{Latin Modern Roman}[OpticalSize=8]}</td>
</tr>
<tr>
<td></td>
<td>\texttt{Latin Modern optical sizes}</td>
</tr>
<tr>
<td></td>
<td>\texttt{\fontspec{Latin Modern Roman}[OpticalSize=12]}</td>
</tr>
<tr>
<td></td>
<td>\texttt{Latin Modern optical sizes}</td>
</tr>
<tr>
<td></td>
<td>\texttt{\fontspec{Latin Modern Roman}[OpticalSize=17]}</td>
</tr>
<tr>
<td></td>
<td>\texttt{Latin Modern optical sizes}</td>
</tr>
</tbody>
</table>

### Example 17: Artificial font transformations.

<table>
<thead>
<tr>
<th>ABCxyz</th>
<th>ABCxyz</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABCxyz</td>
<td>ABCxyz</td>
</tr>
<tr>
<td>ABCxyz</td>
<td>ABCxyz</td>
</tr>
<tr>
<td>ABCxz</td>
<td>ABCxz</td>
</tr>
<tr>
<td>ABCxz</td>
<td>ABCxz</td>
</tr>
<tr>
<td>\texttt{\fontspec{Quattrocento.otf} \emph{ABCxyz} \quad}</td>
<td>\texttt{\fontspec{Quattrocento.otf}[FakeSlant=0.2] ABCxyz}</td>
</tr>
<tr>
<td>\texttt{\fontspec{Quattrocento.otf}[FakeStretch=1.2] ABCxyz}</td>
<td>\texttt{\fontspec{Quattrocento.otf}[FakeBold=1.5] ABCxyz}</td>
</tr>
</tbody>
</table>
If you want the bold shape to be faked automatically, or the italic shape to be slanted automatically, use the \texttt{AutoFakeBold} and \texttt{AutoFakeSlant} features. For example, the following two invocations are equivalent:

\begin{verbatim}
\fontspec[AutoFakeBold=1.5]{Charis SIL}
\fontspec[BoldFeatures={FakeBold=1.5}]{Charis SIL}
\end{verbatim}

If both of the \texttt{AutoFake...} features are used, then the bold italic font will also be faked.

### 6.8 Letter spacing

Letter spacing, or tracking, is the term given to adding (or subtracting) a small amount of horizontal space in between adjacent characters. It is specified with the \texttt{LetterSpace}, which takes a numeric argument, shown in Example \ref{example:letter-space}.

The letter spacing parameter is a normalised additive factor (not a scaling factor); it is defined as a percentage of the font size. That is, for a 10 pt font, a letter spacing parameter of ‘1.8’ will add 0.1 pt between each letter.

This functionality is not generally used for lowercase text in modern typesetting but does have historic precedent in a variety of situations. In particular, small amounts of letter spacing can be very useful, when setting small caps or all caps titles. Also see the OpenType \texttt{Uppercase} option of the \texttt{Letters} feature (\texttt{3.1.7 on page 40}).

Example \ref{example:letter-space}: The LetterSpace feature.

\begin{verbatim}
\fontspec{Didot}
\addfontfeature{LetterSpace=0.0}
\fontspec{Didot}
\addfontfeature{LetterSpace=2.0}
\end{verbatim}
Part IV
OpenType

1 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 3.

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.

1.1 How to select font features

Font features are selected by a series of \texttt{⟨feature⟩=⟨option⟩} selections. Features are (usually) grouped logically; for example, all font features relating to ligatures are accessed by writing \texttt{Ligatures={...}} with the appropriate argument(s), which could be \TeX, \R, etc., as shown below in 3.1.8.

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; \texttt{Numbers={OldStyle,Lining}} doesn’t make much sense because the two options are mutually exclusive, and \TeX will simply use the last option that is specified (in this case using \texttt{Lining} over \texttt{OldStyle}).

If a feature or an option is requested that the font does not have, a warning is given in the console output. As mentioned in Section 3.4 on page 6 these warnings can be suppressed by selecting the [quiet] package option.

31
1.2 How do I know what font features are supported by my fonts?

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

script = ‘latn’
  language = (default)
    features = ‘dtls’ ‘onum’ ‘salt’ ‘ssty’ ‘kern’

script = ‘math’
  language = (default)
    features = ‘dtls’ ‘onum’ ‘salt’ ‘ssty’ ‘kern’
```

I intentionally picked a font above that by design contains 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. After finding the scripts, languages, and features contained within the font, you’ll then need to cross-check the OpenType tags with the ‘logical’ names used by fontspec.

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

```
  otfinfo -f `kpsewhich lmromanunh10-oblique.otf`
```

which results in:

```
salt  Access All Alternates
cpsp  Capital Spacing
dlig  Discretionary Ligatures
frac  Fractions
kern  Kerning
liga  Standard Ligatures
lnum  Lining Figures
onum  Oldstyle Figures
```
2 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 2 on the next page and Table 3 on page 35, respectively. When a script or language is requested that is not supported by the current font, a warning is printed in the console output. See Section 2 on page 62 for methods to create new Script or Language options if required.

Because these font features can change which features are able to be selected for the font, the Script and Language settings are automatically selected by fontspec before all others, and, if Xe\TeX is being used, will specifically select the OpenType renderer for this font, as described in Section 1.2 on page 56.

OpenType fonts can make available different font features depending on the Script and Language chosen. In addition, these settings can also set up their own font behaviour and glyph selection (one example is differences in style between some of the letters in the alphabet used for Bulgarian, Serbian, and Russian). The fontspec feature LocalForms = Off will disable some of these substitutions if desired for some reason. It is important to note that LocalForms = On is a default not of fontspec but of the underlying font shaping engines in both Xe\TeX and Lua\TeX/otfload.

2.1 Script and Language examples

In the examples shown in Example 19, the Code2000 font\footnote{http://www.code2000.net/} 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.

3 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.
Table 2: Defined Scripts for OpenType fonts. Aliased names are shown in adjacent positions marked with red pilcrow symbols (¶).

<table>
<thead>
<tr>
<th>Script</th>
<th>Script</th>
<th>Script</th>
<th>Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adlam</td>
<td>Georgi</td>
<td>Mandaic</td>
<td>Phags-pa</td>
</tr>
<tr>
<td>Ahom</td>
<td>Glagolitic</td>
<td>Manichaean</td>
<td>Phoenician</td>
</tr>
<tr>
<td>Anatolian Hieroglyphs</td>
<td>Gothic</td>
<td>Marchen</td>
<td>Psalter Pahlavi</td>
</tr>
<tr>
<td>Arabic</td>
<td>Grantha</td>
<td>Math</td>
<td>Rejang</td>
</tr>
<tr>
<td>Armenian</td>
<td>Greek</td>
<td>Meitei Mayek</td>
<td>Runic</td>
</tr>
<tr>
<td>Avestan</td>
<td>Gurumuki</td>
<td>Mende Kīlakūti</td>
<td>Samaritan</td>
</tr>
<tr>
<td>Balinese</td>
<td>Hangul      39</td>
<td>Menotic Cursive</td>
<td>Saurashtra</td>
</tr>
<tr>
<td>Barnum</td>
<td>Hangul      39</td>
<td>Menotic Hieroglyphs</td>
<td>Sharada</td>
</tr>
<tr>
<td>Bassa Vah</td>
<td>Hanunoo</td>
<td>Miao</td>
<td>Shavian</td>
</tr>
<tr>
<td>Batak</td>
<td>Hatran</td>
<td>Modī</td>
<td>Siddham</td>
</tr>
<tr>
<td>Bengali</td>
<td>Hebrew</td>
<td>Mongolian</td>
<td>Sing Writing</td>
</tr>
<tr>
<td>Bhoj</td>
<td>Hiragana and Katakana  41</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bopomofo</td>
<td>&amp;Kana</td>
<td>Mro</td>
<td>Shinhala</td>
</tr>
<tr>
<td>Brahmi</td>
<td>Imperial Aramaic</td>
<td>Multani</td>
<td>Sora Sompeng</td>
</tr>
<tr>
<td>Braille</td>
<td>Inscriptional Pahlavi</td>
<td>Musical Symbols</td>
<td>Sumero-Akkadian Cuneiform</td>
</tr>
<tr>
<td>Buginese</td>
<td>Inscriptional Parthian</td>
<td>Myanmar</td>
<td>Sundanese</td>
</tr>
<tr>
<td>Buhid</td>
<td>Javanese</td>
<td>N’ko</td>
<td>Syiolti Nagri</td>
</tr>
<tr>
<td>Byzantine Music</td>
<td>Khmer</td>
<td>N’kō</td>
<td>Syriac</td>
</tr>
<tr>
<td>Canadian Sylabics</td>
<td>Kannada</td>
<td>Nabataean</td>
<td>Tagalog</td>
</tr>
<tr>
<td>Carian</td>
<td>Kayah Li</td>
<td>Neva</td>
<td>Tagbanwa</td>
</tr>
<tr>
<td>Caucasian Albanian</td>
<td>Kharosthi</td>
<td>Ogham</td>
<td>Tai Le</td>
</tr>
<tr>
<td>Chakma</td>
<td>Khmer</td>
<td>Oi Chiki</td>
<td>Tai Lu</td>
</tr>
<tr>
<td>Cham</td>
<td>Khojki</td>
<td>Old Italic</td>
<td>Tai Tham</td>
</tr>
<tr>
<td>Cherokee</td>
<td>Khudawadi</td>
<td>Old Hungarian</td>
<td>Tai Viet</td>
</tr>
<tr>
<td>&amp;CJK</td>
<td>Lao</td>
<td>Old North Arabian</td>
<td>Takri</td>
</tr>
<tr>
<td>&amp;CJK Ideographic</td>
<td>Latin</td>
<td>Old Permic</td>
<td>Tamil</td>
</tr>
<tr>
<td>Coptic</td>
<td>Lepcha</td>
<td>Old Persian Cuneiform</td>
<td>Tangut</td>
</tr>
<tr>
<td>Cyrillic</td>
<td>Limbu</td>
<td>Old South Arabian</td>
<td>Telugu</td>
</tr>
<tr>
<td>Default</td>
<td>Linear A               34</td>
<td>Old Turkic</td>
<td>Thana</td>
</tr>
<tr>
<td>Deseret</td>
<td>Linear B               34</td>
<td>Oniya</td>
<td>Thai</td>
</tr>
<tr>
<td>Devanagari</td>
<td>Lisu</td>
<td>Osage</td>
<td>Tibetan</td>
</tr>
<tr>
<td>Duployan</td>
<td>Lycian</td>
<td>Osmani</td>
<td>Tifinagh</td>
</tr>
<tr>
<td>Egyptian Hieroglyphs</td>
<td>Lydian</td>
<td>Palawv Hmong</td>
<td>Tirhuta</td>
</tr>
<tr>
<td>Elbasan</td>
<td>Mahajani               34</td>
<td>Palymyne</td>
<td>Uguric Cuneiform</td>
</tr>
<tr>
<td>Ethiopic</td>
<td>Malayalam</td>
<td>Pau Cin Hau</td>
<td>Vai</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Warang Citi            34</td>
</tr>
</tbody>
</table>
Table 3: Defined Languages for OpenType fonts. Aliased names are shown in adjacent positions marked with red pilcrow(s).
A brief reference is provided (Table 4 on the next page) but note that this is an incomplete listing — only the ‘enable’ keys are shown, and where alternative interfaces are provided for convenience only the first is shown. (E.g., Numbers=OldStyle is the same as Numbers=Lowercase.)

For completeness, the complete list of OpenType features not provided with a fontspec interface is shown in Table 5 on page 38. 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 GUI 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.

3.1 Tag-based features

3.1.1 Alternates — salt

The Alternate feature, alias StylisticAlternates, is used to access alternate font glyphs when variations exist in the font, such as in Example 20. It uses a numerical selection, starting from zero, that will be different for each font. Note that the Style=Alternate option is equivalent to Alternate=0 to access the default case.

Note that the indexing starts from zero. With the Lua\TeX{} engine, Alternate=Random selects a random alternate.

See Section 1 on page 61 for a way to assign names to alternates if desired.

3.1.2 Character Variants — cvNN

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

For each character that can be varied, it is possible to select among possible options for that particular glyph. For example, in the hypothetical example below, variants are chosen for glyphs ‘4’ and ‘5’, and the trailing \langle n \rangle corresponds to which variety to choose.

\texttt{\fontspec{CV Font}[CharacterVariant={4,5:2}] \& violet}

The numbering is entirely font-specific. Glyph ‘5’ might be the character ‘v’, for example. Character variants are specifically designed not to conflict with each other, so you can enable them individually per character. (Unlike stylistic alternates, say.) Note that the indexing starts from zero.

---

Example 20: The Alternate feature.

\begin{verbatim}
\fontspec{LinLibertine_R.otf}
\textsc{a} \& h \\
\textsc{a} \& h
\addfontfeature{Alternate=0}
\textsc{a} \& h
\end{verbatim}

---

36
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
<th>Feature</th>
<th>Description</th>
<th>Feature</th>
<th>Description</th>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABVM</td>
<td>Diacritics = AboveBase</td>
<td>NLCK</td>
<td>CJKShape = NLC</td>
<td>NUML</td>
<td>Vertical = Numerator</td>
<td>NUMR</td>
<td>Vertical = N</td>
</tr>
<tr>
<td>AFRC</td>
<td>Fractions = Alternate</td>
<td>NUMR</td>
<td>Numbers = Lowercase</td>
<td>ONUM</td>
<td>Numbers = Proportional</td>
<td>ORDN</td>
<td>Vertical = Ordinal</td>
</tr>
<tr>
<td>BLWM</td>
<td>Diacritics = BelowBase</td>
<td>PALT</td>
<td>CharacterWidth = AlternateProportional</td>
<td>PCAP</td>
<td>Letters = PetiteCaps</td>
<td>PKNA</td>
<td>Style = ProportionalKana</td>
</tr>
<tr>
<td>CALT</td>
<td>Contextuals = Alternate</td>
<td>QWID</td>
<td>CharacterWidth = QuarterWidths</td>
<td>RAND</td>
<td>Letters = Random</td>
<td>RLCN</td>
<td>Ligatures = Required</td>
</tr>
<tr>
<td>CASE</td>
<td>Letters = Uppercase</td>
<td></td>
<td></td>
<td>SALT</td>
<td>Alternate = N</td>
<td>SMCP</td>
<td>Letters = SmallCaps</td>
</tr>
<tr>
<td>CLIG</td>
<td>Ligatures = Contextual</td>
<td></td>
<td></td>
<td>SSNN</td>
<td>StylisticSet = N</td>
<td>SSTD</td>
<td>Style = MathScript</td>
</tr>
<tr>
<td>CPSF</td>
<td>Kerning = Uppercase</td>
<td></td>
<td></td>
<td>SUPS</td>
<td>Vertical = Superior</td>
<td>SVSH</td>
<td>Style = Swash</td>
</tr>
<tr>
<td>CSWH</td>
<td>Contextuals = Swash</td>
<td></td>
<td></td>
<td>TNUM</td>
<td>Numbers = Monospaced</td>
<td>TNUM</td>
<td>Numbers = Monospaced</td>
</tr>
<tr>
<td>CVNN</td>
<td>CharacterVariant = N:M</td>
<td></td>
<td></td>
<td>TRAD</td>
<td>CJKShape = Traditional</td>
<td>TNUM</td>
<td>Numbers = Monospaced</td>
</tr>
<tr>
<td>C2PC</td>
<td>Letters = UppercasePetiteCaps</td>
<td></td>
<td></td>
<td>TWID</td>
<td>CharacterWidth = Third</td>
<td>TNUM</td>
<td>Numbers = Monospaced</td>
</tr>
<tr>
<td>C2SC</td>
<td>Letters = UppercaseSmallCaps</td>
<td></td>
<td></td>
<td>UNIC</td>
<td>Letters = Unicase</td>
<td>VALT</td>
<td>Vertical = AlternateMetrics</td>
</tr>
<tr>
<td>D,LIG</td>
<td>Ligatures = Rare</td>
<td></td>
<td></td>
<td>VHAL</td>
<td>Vertical = HalfMetrics</td>
<td>VKNA</td>
<td>Style = VerticalKana</td>
</tr>
<tr>
<td>DNOM</td>
<td>Vertical = Denominator</td>
<td></td>
<td></td>
<td>VRT2</td>
<td>Vertical = RotatedGlyphs</td>
<td>VRLN</td>
<td>Vertical = Kerning</td>
</tr>
<tr>
<td>EXPT</td>
<td>CJKShape = Expert</td>
<td></td>
<td></td>
<td>ZERO</td>
<td>Numbers = SlashedZero</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FALT</td>
<td>Contextuals = LineFinal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FINA</td>
<td>Contextuals = WordFinal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FRAC</td>
<td>Fractions = On</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FWID</td>
<td>CharacterWidth = Full</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HALT</td>
<td>CharacterWidth = AlternateHalf</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HIST</td>
<td>Style = Historic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HKN</td>
<td>Style = HorizontalKana</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HLLG</td>
<td>Ligatures = Historic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HWID</td>
<td>CharacterWidth = Half</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INIT</td>
<td>Contextuals = WordInitial</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ITAL</td>
<td>Style = Italic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JF78</td>
<td>CJKShape = JIS1978</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JF83</td>
<td>CJKShape = JIS1983</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JF90</td>
<td>CJKShape = JIS1990</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JF04</td>
<td>CJKShape = JIS2004</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KERN</td>
<td>Kerning = On</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIGA</td>
<td>Ligatures = Common</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LNUN</td>
<td>Numbers = Uppercase</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOCL</td>
<td>LocalForms = On</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MARK</td>
<td>Diacritics = MarkToBase</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEDI</td>
<td>Contextuals = Inner</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MKMK</td>
<td>Diacritics = MarkToMark</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NALT</td>
<td>Annotation = N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Summary of OpenType features in fontspec, alphabetic by feature tag.
3.1.3 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.

3.1.4 Diacritics

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

3.1.5 Fractions — frac

Activates the construction of ‘vulgar’ fractions using precomposed glyphs and/or subscript and superscript characters from within the font. Coverage will vary by font; see Example 21. Some (Asian fonts predominantly) also provide for the Alternate option.

3.1.6 Kerning — kern

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

<table>
<thead>
<tr>
<th>Feature</th>
<th>Option</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contextuals</td>
<td>Swash</td>
<td>cswh†</td>
</tr>
<tr>
<td></td>
<td>Alternate</td>
<td>cal†</td>
</tr>
<tr>
<td></td>
<td>WordInitial</td>
<td>init†</td>
</tr>
<tr>
<td></td>
<td>WordFinal</td>
<td>fina†</td>
</tr>
<tr>
<td></td>
<td>LineFinal</td>
<td>fall†</td>
</tr>
<tr>
<td></td>
<td>Inner</td>
<td>medi†</td>
</tr>
<tr>
<td></td>
<td>ResetAll</td>
<td></td>
</tr>
</tbody>
</table>

† 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 'Diacritics'.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Option</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diacritics</td>
<td>MarkToBase</td>
<td>mark†</td>
</tr>
<tr>
<td></td>
<td>MarkToMark</td>
<td>mkmk†</td>
</tr>
<tr>
<td></td>
<td>AboveBase</td>
<td>abvm†</td>
</tr>
<tr>
<td></td>
<td>BelowBase</td>
<td>blwm†</td>
</tr>
<tr>
<td></td>
<td>ResetAll</td>
<td></td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Feature</th>
<th>Option</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fractions</td>
<td>On</td>
<td>+frac</td>
</tr>
<tr>
<td></td>
<td>Off</td>
<td>-frac</td>
</tr>
<tr>
<td></td>
<td>Reset</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Alternate</td>
<td>afrc†</td>
</tr>
<tr>
<td></td>
<td>ResetAll</td>
<td></td>
</tr>
</tbody>
</table>

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

Example 21: The Fractions feature.

\setsansfont{Lato}[Fractions=On]
\setmonofont{IBM Plex Mono}[Fractions=On]
\sffamily 1/2 47/11 1/1000 \par
\ttfamily 1/2 47/11

39
Table 9: Options for the OpenType font feature 'Kerning'.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Option</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kerning</td>
<td>On</td>
<td>+kern</td>
</tr>
<tr>
<td></td>
<td>Off</td>
<td>-kern</td>
</tr>
<tr>
<td></td>
<td>Reset</td>
<td></td>
</tr>
<tr>
<td>Uppercase</td>
<td>cpsp</td>
<td>†</td>
</tr>
<tr>
<td></td>
<td>ResetAll</td>
<td></td>
</tr>
</tbody>
</table>

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

As briefly mentioned previously at the end of 3.1.7, the Uppercase option will add a small amount of tracking between uppercase letters, seen in Example 22, which uses the Romande fonts (thanks to Clea F. Rees for the suggestion). The Uppercase option acts separately to the regular kerning controlled by the On/Off options.

3.1.7 Letters

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

Petite caps are smaller than small caps. SmallCaps and PetiteCaps turn lowercase letters into the smaller caps letters, whereas the Uppercase... options turn the capital letters into the smaller caps (good, e.g., for applying to already uppercase acronyms like 'NASA'). This difference is shown in Example 23. 'Unicase' is a weird hybrid of upper and lower case letters.

3.1.8 Ligatures

Ligatures refer to the replacement of two separate characters with a specially drawn glyph for functional or æsthetic reasons. The list of options, of which multiple may be selected at one time, is shown in Table 11. A demonstration with the Linux Libertine fonts is shown in Example 24.

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

Example 22: Adding extra kerning for uppercase letters. (The difference is usually very small.)

UPPERCASE EXAMPLE
UPPERCASE EXAMPLE
UPPERCASE EXAMPLE
UPPERCASE EXAMPLE

\fontspec{RomandeADFStd-DemiBold.otf}
UPPERCASE EXAMPLE \%
\addfontfeature{Kerning=Uppercase}
UPPERCASE EXAMPLE
Table 10: Options for the OpenType font feature ‘Letters’.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Option</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Letters</td>
<td>SmallCaps</td>
<td>smcp</td>
</tr>
<tr>
<td></td>
<td>PetiteCaps</td>
<td>pcap</td>
</tr>
<tr>
<td></td>
<td>UppercaseSmallCaps</td>
<td>c2sc</td>
</tr>
<tr>
<td></td>
<td>UppercasePetiteCaps</td>
<td>c2pc</td>
</tr>
<tr>
<td></td>
<td>Unicase</td>
<td>unic</td>
</tr>
<tr>
<td></td>
<td>ResetAll</td>
<td></td>
</tr>
</tbody>
</table>

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

Example 23: Small caps from lowercase or uppercase letters.

\[\fontspec\{texgyreadventor-regular.otf\}\{Letters=SmallCaps\} \]
\[\fontspec\{texgyreadventor-regular.otf\}\{Letters=UppercaseSmallCaps\}\]

Mapping feature (see Section 1.1 on page 56) but for consistency Ligatures=TeX will perform the same function as Mapping=tex-text.

3.1.9 Localised Forms — locl

This feature enables and disables glyph substitutions, etc., that are specific to the Language selected in the font. This feature is automatically activated by default when present, so it should not be generally necessary to use LocalForms = On. In certain scenarios it may be important to turn it Off (although nothing specifically springs to mind).

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

<table>
<thead>
<tr>
<th>Feature</th>
<th>Option</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ligatures</td>
<td>Required</td>
<td>rlig</td>
</tr>
<tr>
<td></td>
<td>Common</td>
<td>liga</td>
</tr>
<tr>
<td></td>
<td>Contextual</td>
<td>clig</td>
</tr>
<tr>
<td></td>
<td>Rare/Discretionary</td>
<td>dlig</td>
</tr>
<tr>
<td></td>
<td>Historic</td>
<td>hlig</td>
</tr>
<tr>
<td></td>
<td>TeX</td>
<td>tlig</td>
</tr>
<tr>
<td></td>
<td>ResetAll</td>
<td></td>
</tr>
</tbody>
</table>

† These feature options can be disabled with ..Off variants, and reset to default state (neither explicitly on nor off) with ..Reset.
Example 24: An example of the Ligatures feature.

\def\test#1#2{\%
 #2 $\to$ \{\addfontfeature{#1} #2\}\}
\fontspec{LinLibertine_R.otf}
\test{Ligatures=Historic}{strict}
\test{Ligatures=Rare}{wurtzite}
\test{Ligatures=CommonOff}{firefly}

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

<table>
<thead>
<tr>
<th>Feature</th>
<th>Option Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>LocalForms</td>
<td>= On +locl</td>
</tr>
<tr>
<td></td>
<td>Off -locl</td>
</tr>
<tr>
<td></td>
<td>Reset</td>
</tr>
</tbody>
</table>

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

3.1.10 Numbers

The Numbers feature defines how numbers will look in the selected font, accepting options shown in Table 13.

The synonyms Uppercase and Lowercase are equivalent to Lining and OldStyle, respectively. The differences have been shown previously in Section 2 on page 20. 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 uppercse 'O', shown in Example 25.

The Arabic option (with tag anum) maps regular numerals to their Arabic script or Persian equivalents based on the current Language setting (see Section 2 on page 33). This option is based on a Lua\TeX\ feature of the luaotfload package, not an OpenType feature. (Thus, this feature is unavailable in Xe\TeX.) This feature should be considered deprecated; while there are no plans to remove it from this package, if its support is dropped from the font loader it could disappear from fontspec with little notice.

Example 25: The effect of the SlashedZero option.

\fontspec[Numbers=Lining]{texgyrebonum-regular.otf}
0123456789
\fontspec[Numbers=SlashedZero]{texgyrebonum-regular.otf}
0123456789
Table 13: Options for the OpenType font feature 'Numbers'.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Option</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numbers</td>
<td>Uppercase</td>
<td>inum†</td>
</tr>
<tr>
<td></td>
<td>Lowercase</td>
<td>onum†</td>
</tr>
<tr>
<td></td>
<td>Lining</td>
<td>num †</td>
</tr>
<tr>
<td></td>
<td>OldStyle</td>
<td>onum†</td>
</tr>
<tr>
<td></td>
<td>Proportional</td>
<td>pnum†</td>
</tr>
<tr>
<td></td>
<td>Monospaced</td>
<td>tnum†</td>
</tr>
<tr>
<td></td>
<td>SlashedZero</td>
<td>zero†</td>
</tr>
<tr>
<td></td>
<td>Arabic</td>
<td>anum†</td>
</tr>
<tr>
<td>ResetAll</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

3.1.11 Ornament — ornm

Ornaments are selected with the Ornament feature (OpenType feature ornm), selected numerically such as for the Annotation feature.

3.1.12 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 26 shows an example of a font feature that involves glyph substitution for particular letters within an alphabet. Other options in these categories operate in similar ways, with the choice of how particular substitutions are organised with which feature largely up to the font designer.

The Uppercase option is designed to select various uppercase forms for glyphs such as accents and dashes, such as shown in Example 27; note the raised position of the hyphen to better match the surrounding letters. It will (probably) not actually map letters to uppercase. This option used to be selected under the Letters feature, but moved here as it generally does not actually affect the letters themselves. The Kerning feature also contains an Uppercase option, which adds a small amount of spacing in between letters (see 3.1.6 on page 38).

In other features, larger breadths of changes can be seen, covering the style of an entire alphabet. See Example 28; here, the Italic option affects the Latin text and the Ruby option the Japanese.

Example 26: Example of the Alternate option of the Style feature.

\fontspec{Quattrocento.otf}
M Q W
\addfontfeature{Style=Alternate}
M Q W

\addfontfeature{Style=Alternate}
M Q W

---

8If you want automatic uppercase letters, look to \E\LaTeX’s \MakeUppercase command.
Table 14: Options for the OpenType font feature ‘Style’.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Option</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Style</td>
<td>Alternate</td>
<td>salt†</td>
</tr>
<tr>
<td></td>
<td>Italic</td>
<td>ital†</td>
</tr>
<tr>
<td></td>
<td>Ruby</td>
<td>ruby†</td>
</tr>
<tr>
<td></td>
<td>Swash</td>
<td>swsh†</td>
</tr>
<tr>
<td></td>
<td>Cursive</td>
<td>curs†</td>
</tr>
<tr>
<td></td>
<td>Historic</td>
<td>hist†</td>
</tr>
<tr>
<td></td>
<td>Titling</td>
<td>titl†</td>
</tr>
<tr>
<td></td>
<td>HorizontalKana</td>
<td>hkna†</td>
</tr>
<tr>
<td></td>
<td>VerticalKana</td>
<td>vkna†</td>
</tr>
<tr>
<td></td>
<td>ResetAll</td>
<td></td>
</tr>
</tbody>
</table>

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

Example 27: An example of the Uppercase option of the Style feature.

\fontspec{LinLibertine_R.otf}
\addfontfeature{Style=Uppercase}

Example 28: Example of the Italic and Ruby options of the Style feature.

\fontspec{Hiragino Mincho Pro}
\addfontfeature{Style={Italic, Ruby}}
Note the difference here between the default and the horizontal style kana in Example 29: the horizontal style is slightly wider.

### 3.1.13 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 font⁹ are shown in Example 30 and Example 31; 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 1 on page 61 for a way to assign names to stylistic sets, which should be done on a per-font basis.

### 3.1.14 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 32.

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

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

### 3.2 CJK features

This section summarises the features which are largely intending for Chinese, Korean, and Japanese typesetting.

---

⁹http://junicode.sf.net

---

Example 29: Example of the HorizontalKana and VerticalKana options of the Style feature.

```
\fontspec{Hiragino Mincho Pro} \kana \\ {\addfontfeature{Style=HorizontalKana} \kana } \\ {\addfontfeature{Style=VerticalKana} \kana }
```
Example 30: Insular letterforms, as used in medieval Northern Europe, for the Junicode font accessed with the StylisticSet feature.

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

---

Example 31: Enlarged minuscules (capital letters remain unchanged) for the Junicode font, accessed with the StylisticSet feature.

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

---

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

<table>
<thead>
<tr>
<th>Feature</th>
<th>Option</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>VerticalPosition  = Superior</td>
<td>sups †</td>
<td></td>
</tr>
<tr>
<td>Inferior</td>
<td>subs †</td>
<td></td>
</tr>
<tr>
<td>Numerator</td>
<td>numr †</td>
<td></td>
</tr>
<tr>
<td>Denominator</td>
<td>dnom †</td>
<td></td>
</tr>
<tr>
<td>ScientificInferior</td>
<td>sinf †</td>
<td></td>
</tr>
<tr>
<td>Ordinal</td>
<td>ordn †</td>
<td></td>
</tr>
<tr>
<td>ResetAll</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

---

Example 32: The VerticalPosition feature.

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

---

46
3.2.1 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 33. Note that the indexing starts from zero.

3.2.2 Character width

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

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

The same situation occurs with numbers, which are provided in increasingly illegible compressed forms seen in Example 35.

3.2.3 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.

3.2.4 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>
<thead>
<tr>
<th>Feature</th>
<th>Option</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>CharacterWidth = Proportional</td>
<td>pwid †</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Full</td>
<td>fwid †</td>
</tr>
<tr>
<td></td>
<td>Half</td>
<td>hwid †</td>
</tr>
<tr>
<td></td>
<td>Third</td>
<td>twid †</td>
</tr>
<tr>
<td></td>
<td>Quarter</td>
<td>qwid †</td>
</tr>
<tr>
<td></td>
<td>AlternateProportional</td>
<td>palt †</td>
</tr>
<tr>
<td></td>
<td>AlternateHalf</td>
<td>halt †</td>
</tr>
<tr>
<td>ResetAll</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

† These feature options can be disabled with .Off variants, and reset to default state (neither explicitly on nor off) with .Reset.
Example 33: Annotation forms for OpenType fonts.

\begin{verbatim}
\[1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \]
(1) \[2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \]
(1) \[2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \]
\end{verbatim}

Example 34: Proportional or fixed width forms.

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

Example 35: Numbers can be compressed significantly.

\begin{verbatim}
\fontspec[Renderer=AAT]{Hiragino Mincho Pro}{\addfontfeature{CharacterWidth=Full}---12321---}{\addfontfeature{CharacterWidth=Half}---123454321---}{\addfontfeature{CharacterWidth=Third}---123456787654321---}{\addfontfeature{CharacterWidth=Quarter}---12345678900987654321---}
\end{verbatim}

Example 36: Different standards for CJK ideograph presentation.

\begin{verbatim}
\fontspec{Hiragino Mincho Pro}{\addfontfeature{CJKShape=Traditional}\text}{\addfontfeature{CJKShape=NLC}\text}{\addfontfeature{CJKShape=Expert}\text}
\end{verbatim}
Table 17: Options for the OpenType font feature ‘CJKShape’.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Option</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>CJKShape</td>
<td>Traditional trad</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Simplified   smpl</td>
<td></td>
</tr>
<tr>
<td></td>
<td>JIS1978     jp78</td>
<td></td>
</tr>
<tr>
<td></td>
<td>JIS1983     jp83</td>
<td></td>
</tr>
<tr>
<td></td>
<td>JIS1990     jp90</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Expert       expt</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NLC          nlck</td>
<td></td>
</tr>
</tbody>
</table>

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

Table 18: Options for the OpenType font feature ‘Vertical’.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Option</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertical</td>
<td>RotatedGlyphs</td>
<td>vrt2†</td>
</tr>
<tr>
<td></td>
<td>AlternatesForRotation</td>
<td>vrtr†</td>
</tr>
<tr>
<td></td>
<td>Alternates</td>
<td>vert†</td>
</tr>
<tr>
<td></td>
<td>KanaAlternates</td>
<td>vkna†</td>
</tr>
<tr>
<td></td>
<td>Kerning</td>
<td>vkrn†</td>
</tr>
<tr>
<td></td>
<td>AlternateMetrics</td>
<td>valt†</td>
</tr>
<tr>
<td></td>
<td>HalfMetrics</td>
<td>vhal†</td>
</tr>
<tr>
<td></td>
<td>ProportionalMetrics</td>
<td>vpal†</td>
</tr>
<tr>
<td></td>
<td>ResetAll</td>
<td></td>
</tr>
</tbody>
</table>

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

Commands for accents and symbols (‘encodings’)

The functionality described in this section is experimental.

In the pre-Unicode era, significant work was required by \texttt{\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 \texttt{babel} and \texttt{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, \texttt{fontspec} can leverage the font encoding technology built into \texttt{\LaTeX}x2 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 \texttt{\LaTeX}x commands rather than Unicode input; for example, typing \texttt{\textcyr{e}} instead of \texttt{\euro} or \texttt{\textcopyright} instead of \texttt{\copyright} in the source file.

The most widely-used encoding in \texttt{\LaTeX}x was \texttt{T1} with companion ‘\texttt{TS1}’ symbols provided by the \texttt{textcomp} package. These encodings provided glyphs to typeset text in a variety of western European languages. As with most legacy \texttt{\LaTeX}x input methods, accents and symbols were input using encoding-dependent commands such as \texttt{\textcyr{e}} as described above. As of 2017, in \texttt{\LaTeX}x on \texttt{Xe\LaTeX} and \texttt{Lua\LaTeX}, the default encoding is \texttt{TU}, which uses Unicode for input and output. The \texttt{TU} encoding provides appropriate encoding-dependent definitions for input commands to match the coverage of the \texttt{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, \texttt{fontspec} provides an interface for a more extensible system.

1 A new Unicode-based encoding from scratch

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

\begin{verbatim}
\DeclareTextSymbol{\CYRIE}{OT2}{5}
\DeclareTextSymbol{\CYRDJE}{OT2}{6}
\DeclareTextSymbol{\CYRTSHE}{OT2}{7}
\DeclareTextSymbol{\cyrnje}{OT2}{8}
\end{verbatim}
To recreate this encoding in a form suitable for fontspec, create a new file named, say, fontrange-cyr.def and populate it with

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

...  

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

\EncodingSymbol{\CYRIE}{"0404}  

...  

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

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

Then follow it up with a font loading call such as

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

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

2 Adjusting a pre-existing encoding

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

When adding symbols, etc., simply write

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


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

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

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

Provided that you use the command \texttt{\textinterrobang} to typeset this symbol, it will appear in fonts with the default encoding, while in any font loaded with the \texttt{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 \TeX as:

\texttt{\EncodingAccent{d}{"0323}}

For fonts with a missing (or poorly-spaced) “0323 accent glyph, the ‘traditional’ \TeX fake accent construction could be used instead:

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

This would be set up in a document as such:

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

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

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

To reiterate from above, typing this input with Unicode text (‘kalitaṃ’) will bypass this encoding mechanism and you will receive only what is contained literally within the font.
3 Summary of commands

The \TeX kernel provides the following font encoding commands suitable for Unicode encodings:

\begin{verbatim}
\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)}
\end{verbatim}

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:

\begin{verbatim}
\EncodingCommand{(command)}{(num)}{(default)}{(code)}
\EncodingAccent{(command)}{(code)}
\EncodingSymbol{(command)}{(code)}
\EncodingComposite{(command)}{(letter)}{(slot)}
\EncodingCompositeCommand{(command)}{(letter)}{(code)}
\UndeclareSymbol{(command)}
\UndeclareAccent{(command)}
\UndeclareCommand{(command)}
\UndeclareComposite{(command)}{(letter)}
\end{verbatim}
Part VI
Lua\TeX-only font features

1 Different font technologies and shapers

Lua\TeX does not directly support any font rendering technologies out of the box, it requires additional functionality to be added to properly support and control technologies such as OpenType.

Using the \texttt{Renderer} feature, there are a number of options that \texttt{fontspec} can pass to the engine to control which font technology is being used. Pre-2019, there were two options provided by \texttt{luaotfload} that generally did not require user intervention.

- \texttt{Renderer = Node}: the default ‘mode’ for typesetting OpenType fonts.
- \texttt{Renderer = Base}: a simplified mode useful only in a limited number of situations such as mathematics typesetting.

From 2019 the possibility of using the Harfbuzz text shaping engine within Lua\TeX has been developed by Khaled Hosny. When running a suitable Lua\TeX engine with Harfbuzz support, \texttt{fontspec} provides the following options:

- \texttt{Renderer = Harfbuzz}: use the Harfbuzz engine without an explicit ‘shaper’.
- \texttt{Renderer = OpenType}: use the Harfbuzz engine with the OpenType shaper.
- \texttt{Renderer = AAT}: use the Harfbuzz engine with the AAT shaper.
- \texttt{Renderer = Graphite}: use the Harfbuzz engine with the Graphite shaper.
- \texttt{Renderer = ⟨foo⟩}: use the Harfbuzz engine with the ⟨foo⟩ shaper.

Support for the Harfbuzz renderer is preliminary and may be improved over time. Please treat the interface for Harfbuzz fonts as subject to change.

2 Custom font features

Lua\TeX, via the \texttt{luaotfload} package, allows the definition and re-definition of custom OpenType features for a selected font. This facility is particularly useful to implement custom substitutions or to disable unwanted but not all ligatures.

Figure 1 shows an minimal example of this type of functionality. This example creates a new OpenType feature, \texttt{oneb}, which substitutes the glyph when typesetting ’1’ for the named glyph \texttt{one.as@1}. The glyph names are font specific and can be interrogated with third-party software such as FontForge.

A third-party collection of additional examples are maintained in the repository \texttt{fonts-in-luatex}\textsuperscript{10}. These examples are intended to correct or adjust font features in a range of commercial fonts and provide a good introduction to some of the possibilities that Lua\TeX affords.

Please refer to the Lua\TeX/luaotfload documentation for more details.

\textsuperscript{10}
\url{https://github.com/mewtant/fonts-in-luatex}
Figure 1: An example of custom font features.

\documentclass{article}
\usepackage{fontspec}
\directlua{
  fonts.handlers.otf.addfeature {
    name = "oneb",
    type = "substitution",
    data = {
      ["i"] = "one.ss@1",
    }
  }
}
\setmainfont{Vollkorn-Regular.otf}[RawFeature=+oneb]
\begin{document}
1234567890
\end{document}
Part VII
Fonts and features with \textit{XeLaTeX}

1 \textit{XeLaTeX}-only font features

The features described here are available for any font selected by \texttt{fontspec}.

1.1 Mapping

The \texttt{Mapping} feature enables a \textit{XeLaTeX} text-mapping scheme, with an example shown in \textit{Example 37}. Only one mapping can be active at a time and a second call to \texttt{Mapping} will override the first. Using the \texttt{tex-text} mapping is also equivalent to writing \texttt{Ligatures=TeX}. The use of the latter syntax is recommended for better compatibility with \textit{LuaLaTeX} documents.

1.2 Different font technologies: \texttt{AAT}, \texttt{OpenType}, and \texttt{Graphite}

Note that from 2020 it appears that \textit{XeLaTeX} can no longer support \texttt{AAT} fonts in macOS. \textit{XeLaTeX} supports three rendering technologies for typesetting, selected with the \texttt{Renderer} font feature. The first, \texttt{AAT}, is that provided only by macOS. The second, \texttt{OpenType}, is an open source OpenType interpreter. It provides greater support for OpenType features, notably contextual arrangement, over \texttt{AAT}. The third is \texttt{Graphite}, which is an alternative to OpenType with particular features for less-common languages and the capability for more powerful font options. Features for \texttt{OpenType} have already been discussed in \textit{IV} on page 31; \texttt{Graphite} and \texttt{AAT} features are discussed later in \textit{Section 2 on the next page} and \textit{Section 3 on page 58}.

Unless you have a particular need, the \texttt{Renderer} feature is rarely explicitly required: for \texttt{OpenType} fonts, the \texttt{OpenType} renderer is used automatically, and for \texttt{AAT} fonts, \texttt{AAT} is chosen by default. Some fonts, however, will contain font tables for multiple rendering technologies, such as the Hiragino Japanese fonts distributed with macOS, and in these cases one over the other may be preferred.

Among some other font features only available through a specific renderer, \texttt{OpenType} provides for the \texttt{Script} and \texttt{Language} features, which allow different font behaviour for different alphabets and languages; see \textit{Section 2 on page 33} for the description of these features. \textit{Because these font features can change which features are able to be selected for the font instance, they are selected by \texttt{fontspec} before all others and will automatically and without warning select the \texttt{OpenType} renderer.}

\begin{center}
\textbf{Example 37:} \textit{XeLaTeX}'s \texttt{Mapping} feature.
\end{center}

\begin{verbatim}
"¡A small amount of—text!" \fontspec{texgyrepagella-regular.otf}[Mapping=tex-text]
\text{"¡A small amount of—text!"} \end{verbatim}
1.3 Optical font sizes

Multiple Master fonts are parameterised over orthogonal font axes, allowing continuous selection along such features as weight, width, and optical size. Whereas an OpenType font will have only a few separate optical sizes, a Multiple Master font’s optical size can be specified over a continuous range. Unfortunately, this flexibility makes it harder to create an automatic interface through \LaTeX, and the optical size for a Multiple Master font must always be specified explicitly.

\begin{verbatim}
\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  \\
\end{verbatim}

1.4 Vertical typesetting

\LaTeX provides for vertical typesetting simply with the ability to rotate the individual glyphs as a font is used for typesetting, as shown in Example 38.

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 \LaTeX documentation.

2 The Graphite renderer

Since the Graphite renderer is designed for less common scripts and languages, usually with specific or unique requirements, Graphite features are not standard across fonts.

Currently fontspec does not support a convenient interface to select Graphite font features and all selection must be done via ‘raw’ font feature selection.

Here’s an example:

\begin{verbatim}
\fontspec{Charis SIL}[Renderer=Graphite, 
  RawFeature={Uppercase Eng alternates=Large eng on baseline}]
\end{verbatim}

Example 38: Vertical typesetting.

共産主義者は

\begin{verbatim}
\fontspec{Hiragino Mincho Pro}
\verbatim
\fontspec{Hiragino Mincho Pro}[Renderer=AAT,Vertical=RotatedGlyphs]
\rotatebox{-90}{\verbatim} % requires the graphicx package
\end{verbatim}
Here’s another:
\fontspec{AwamiNastaliq-Regular.ttf}[Renderer=Graphite] ^^^^06b5
\addfontfeature{RawFeature={Lam with V=V over bowl}} ^^^^06b5

3 macOS’s AAT fonts

Warning! Xe\LaTeX’s implementation on macOS 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
Xe\LaTeX’s completes its transition to a cross-platform–only application. All examples
in this section have now been removed.

macOS’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 Xe\LaTeX (due to its pedigree
on macOS in the first place) and was the first font format supported by fontspec. A number
of fonts distributed with macOS are still in the AAT format, such as ‘Skia’.

3.1 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 combina-
tion 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 fea-
ture. Notice also that the old \TeX trick of splitting up a ligature with an empty brace pair does
not work in Xe\LaTeX; you must use a 0 pt kern or \hbox (e.g., \null) to split the characters up if
you do not want a ligature to be performed (the usual examples for when this might be desired
are words like ‘shelffull’).

3.2 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.

3.3 Numbers

The Numbers feature defines how numbers will look in the selected font. For AAT fonts, they
may be a combination of Lining or OldStyle and Proportional or Monospaced (the latter
is good for tabular material). The synonyms Uppercase and Lowercase are equivalent to
Lining and OldStyle, respectively. The differences have been shown previously in Section 2
on page 20.
3.4 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 ??), LineInitial, LineFinal, and Inner (Example ??, also called ‘non-final’ sometimes). As non-exclusive selectors, like the ligatures, you can turn them off by prefixing their name with No.

3.5 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.

The realscripts package redefines the \textsubscript and \textsuperscript commands to use the above font features, including for use in footnote labels.

3.6 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.

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.

3.7 Variants

The Variant feature takes a single numerical input for choosing different alphabetic shapes. See Section 1 on page 61 for a way to assign names to variants, which should be done on a per-font basis.

3.8 Alternates

Selection of Alternates again must be done numerically. See Section 1 on page 61 for a way to assign names to alternates, which should be done on a per-font basis.

3.9 Style

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

Typical examples for these features are shown in 3.1.12.

\footnote{Ruby’ refers to a small optical size, used in Japanese typography for annotations.}
3.10 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.

3.11 Character width

See 3.2.2 on page 47 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.

3.12 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 macOS included ‘ø’ etc. as shorthand for writing ‘Ø’ under the label of the Diacritics feature. If you come across such fonts, you’ll want to turn this feature off (imagine typing hello/goodbye and getting ‘helløgoodbye’ instead!) by decomposing the two characters in the diacritic into the ones you actually want. I recommend using the proper \LaTeX input conventions for obtaining such characters instead.

3.13 Annotation

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

This chapter describes the current interfaces and hooks that use fontspec for various macro programming purposes.

1 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 Xe\TeX file AAT-info.tex to obtain the code numbers. See Example 39.

\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}{-loc1}

In case the above commands do not accommodate the desired font feature (perhaps a new Xe\TeX 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 something like the following:

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

Example 39: Assigning new AAT features.

This is Xe\TeX by Jonathan Kew.
The advantage to using the \texttt{newAATfeature} and \texttt{newopentypefeature} commands instead of \texttt{newfontfeature} is that they check if the selected font actually contains the desired font feature at load time. By contrast, \texttt{newfontfeature} will not give a warning for improper input.

2 Defining new scripts and languages

While the scripts and languages listed in Table 2 and Table 3 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 \texttt{newfontscript} and \texttt{newfontlanguage} commands. For example,

\begin{verbatim}
newfontscript{Arabic}{arab}
newfontlanguage{Zulu}{ZUL}
\end{verbatim}

The first argument is the fontspec name, the second the OpenType tag. The advantage to using these commands rather than \texttt{newfontfeature} (see Section 1 on the preceding page) is the error-checking that is performed when the script or language is requested.

Both commands accept a comma-separated list of OpenType tags in order of preference. This permits, for example, supporting both new and old versions of a language tag with a common user interface:

\begin{verbatim}
newfontlanguage{Turkish}{TRK,TUR}
\end{verbatim}

Here, a font that is requested with Script=Turkish will first be checked for the OpenType language tag TRK, which will be selected if available. If not available, the TUR tag will be queried and used if possible as a fallback.

3 Going behind fontspec’s back

Expert users may wish not to use fontspec’s feature handling at all, while still taking advantage of its L\TeX{} font selection conveniences. The \texttt{RawFeature} font feature allows font feature selection using a literal feature selection string if you happen to have the OpenType feature tag memorised. More importantly, this can be used to enable features for which fontspec does not yet have a user interface to.

Multiple features can either be included in a single declaration:

\begin{verbatim}
[RawFeature=+smcp;+onum]
\end{verbatim}

or with multiple declarations:

\begin{verbatim}
[RawFeature=+smcp, RawFeature=+onum]
\end{verbatim}

Note that there is no error-checking when using \texttt{RawFeature}. Where a fontspec interface exists to a feature it is generally better to use it. If the font lacks the feature or if it would clash with another feature, fontspec will attempt to warn and/or resolve the issues.

---

<table>
<thead>
<tr>
<th>Example 40: Using raw font features directly.</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{\fontspec{texgyrepagella-regular.otf}[RawFeature=+smcp]}</td>
</tr>
<tr>
<td>\texttt{Pagella small caps}</td>
</tr>
</tbody>
</table>

62
4 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 41.

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

\aliasfontfeatureoption

If you wish to change the name of a font feature option, it can be aliased to another with the command \aliasfontfeatureoption{⟨font feature⟩}{⟨existing name⟩}{⟨new name⟩}, such as shown in Example 42.

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 550899BB 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=550899BB}

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

5 Programming interface

5.1 Variables

\l_fontspec_family_tl
\l_fontspec_font
\g_fontspec_encoding_tl

In some cases, it is useful to know what the \TeX font family of a specific fontspec font is. After a \fontspec-like command, this is stored inside the \l_fontspec_family_tl macro. Otherwise, \TeX’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 \TeX NFSS commands may also need to know what the default font encoding is. Since this has changed from EU1/EU2 to TU, it is best to use the variable \g_fontspec_encoding_tl instead.

5.2 Functions for loading new fonts and families

\fontspec_gset_family:Nnn
\fontspec_set_family:Nnn

#1 : \TeX family
#2 : fontspec features
#3 : font name

Defines a new nsrs family from given ⟨features⟩ and ⟨font⟩, and stores the family name in the variable ⟨family⟩. This font family can then be selected with standard \TeX commands

---

Example 41: Renaming font features.

<table>
<thead>
<tr>
<th>Roman Letters And Swash</th>
<th>\aliasfontfeature{ItalpFeatures}{IF}</th>
<th>\fontspec{Hoefler Text}{IF = {Alternate=1}}</th>
<th>Roman Letters \lttshape And Swash</th>
</tr>
</thead>
</table>

63
Example 42: Renaming font feature options.

\aliasfontfeature{VerticalPosition}{Vert Pos}
\aliasfontfeatureoption{VerticalPosition}{ScientificInferior}{Sci Inf}
\fontspec{LinLibertine_R.otf}[Vert Pos=Sci Inf]

Scientific Inferior: 12345

\fontfamily{⟨family⟩}\selectfont. See the standard fontspec user commands for applications of this function.

(End definition for \fontspec_gset_family:Nnn and \fontspec_set_family:Nnn. These functions are documented on page ??.)

\fontspec_gset_fontface:NNnn \fontspec_set_fontface:NNnn

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

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

(End definition for \fontspec_gset_fontface:NNnn and \fontspec_set_fontface:NNnn. These functions are documented on page ??.)

5.3 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.

5.3.1 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}

(End definition for \fontspec_font_if_exist:nTF. This function is documented on page ??.)

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

\fontspec_if_fontspec_font:TF

Test whether the currently selected font has been loaded by fontspec.

(End definition for \fontspec_if_fontspec_font:TF. This function is documented on page ??.)

\fontspec_if_opentype:TF

Test whether the currently selected font is an OpenType font. Always true for Lua\LaTeX fonts.
Test whether the currently selected font has a ‘small caps’ face to be selected with \scshape or similar. Note that testing whether the font has the Letters=SmallCaps font feature is sufficient but not necessary for this command to return true, since small caps can also be loaded from separate font files. The logic of this command is complicated by the fact that fontspec will merge shapes together (for italic small caps, etc.).

5.3.2 Availability of features

Test whether the currently selected font contains the AAT feature (#1,#2).

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.

Test whether the currently selected font with raw OpenType script tag #1 and raw OpenType language tag #2 contains the raw OpenType feature tag #3. E.g.: \fontspec_if_feature:nnnTF {latn} {ROM} {pnum} {True} {False}. Returns false if the font is not loaded by fontspec or is not an OpenType font.

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

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

5.3.3 Currently selected features

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.

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

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