

The *fontspec* package

WILL ROBERTSON

2006/12/24 v1.14

Contents

1	Introduction	2	6.6	Contextuals	18
1.1	Usage	2	6.7	Vertical position	19
1.2	About this manual	3	6.8	Fractions	19
2	Brief overview	4	6.9	Variants	20
3	Font selection	4	6.10	AAT Alternates	21
3.1	Default font families	5	6.11	Style	21
3.2	Font instances for efficiency	5	6.12	Diacritics	22
3.3	Arbitrary bold/italic/small caps fonts	6	6.13	Kerning	22
3.4	Math(s) fonts	6	6.14	CJK shape	23
3.5	External fonts	7	6.15	Character width	23
3.6	Miscellaneous font selecting details	8	6.16	Annotation	24
4	Selecting font features	8	6.17	Vertical typesetting	24
4.1	Default settings	8	6.18	AAT & Multiple Master font axes	25
4.2	Changing the currently selected features	8	6.19	OpenType scripts and languages	25
4.3	Priority of feature selection	9	7	Defining new features	27
4.4	Different features for different font shapes	9	7.1	Renaming existing features & options	29
4.5	Different features for different font sizes	10	7.2	Going behind fontspec's back	30
5	Font independent options	11	I	fontspec.sty	31
5.1	Scale	11	8	Implementation	31
5.2	Mapping	11	8.1	Bits and pieces	31
5.3	Colour	12	8.2	Option processing	32
5.4	Interword space	12	8.3	Packages	32
5.5	Post-punctuation space	13	8.4	Encodings	32
5.6	Letter spacing	13	8.5	User commands	33
5.7	The hyphenation character	13	8.6	Internal macros	37
6	Font-dependent features	14	8.7	keyval definitions	49
6.1	Different font technologies: AAT and ICU	15	8.8	Italic small caps	64
6.2	Optical font sizes	15	8.9	Selecting maths fonts	65
6.3	Ligatures	16	8.10	Finishing up	68
6.4	Letters	17	II	fontspec.cfg	69
6.5	Numbers	18	III	fontspec-example.ltx	69

1 Introduction

With the introduction of Jonathan Kew's Xe_ΛTeX,¹ users can now easily access system-wide fonts directly in a TeX variant, providing a best of both worlds environment. Xe_ΛTeX eliminates the need for all those files required for installing fonts (.tfm, .vf, .map, ...) and provides an easy way to select fonts in Plain TeX: `\font\tenrm="Times New Roman" at 10pt`.

Before `fontspec`, it was still necessary to write cumbersome font definition files for L^ATeX, since the NFSS had a lot more going on behind the scenes to allow easy commands like `\emph` or `\bfseries`.

This package provides a completely automatic way to select font families in L^ATeX for arbitrary fonts. Furthermore, it allows very flexible control over the selection of advanced font features such as number case and fancy ligatures (and many more!) present in most modern fonts.

1.1 Usage

For basic use, no package options are required:

```
\usepackage{fontspec}% provides font selecting commands
\usepackage{xunicode}% provides unicode character macros
\usepackage{xltextra} % provides some fixes/extras
```

Ross Moore's `xunicode` package is highly recommended, as it provides access L^ATeX's various methods for accessing extra characters and accents (for example, `\%`, `\$`, `\textbullet`, `\"u`, and so on), plus many more unicode characters.

The `xltextra` package adds a couple of general improvements to L^ATeX under Xe_ΛTeX; it also provides the `\XeTeX` macro to typeset the Xe_ΛTeX logo.

The babel package is not really supported! Especially Vietnamese, Greek, and Hebrew at least might not work correctly, as far as I can tell. There's a better chance with Cyrillic and Latin-based languages, however—`fontspec` ensures at least that fonts should load correctly, but hyphenation and other matters aren't guaranteed.

The rest of this section documents `fontspec`'s package options, which are (briefly):

```
cm-default Don't load the Latin Modern fonts;
no-math Don't change any maths fonts;
no-config Don't load fontspec.cfg; and,
quiet Output fontspec warnings in the log file rather than the console output.
```

1.1.1 Latin Modern defaults

`fontspec` defines a new L^ATeX font encoding for its purposes to allow the Latin Modern fonts to be used by default. This has three implications:

★ v1.12: New!

¹<http://scripts.sil.org/xetex>

1. Unicode fonts are loaded by default; it didn't make sense to have the legacy Computer Modern fonts in the Unicode-enabled Xe_{La}TeX.
2. If you don't have the Latin Modern OpenType fonts installed, you might want to consider doing so.
3. `fontspec` also requires the `euenc` package² to be installed.

Another package option is provided for controlling this behaviour: `[cm-default]` will ignore the Latin Modern fonts and go about things as it used to. Use this option if you don't have the Latin Modern fonts installed or you (Mac-specifically) want to use the 'default TeX font' without using the `xdvipdfmx` driver.

1.1.2 Maths 'fiddling'

★ v1.14: New!

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

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

1.1.3 Configuration

If you wish to customise any part of the `fontspec` interface (see later in this manual, Section 7 on page 27 and Section 7.1), this should be done by creating your own `fontspec.cfg` file,³ which will be automatically loaded if it is found by Xe_{La}TeX. Either place it in the same folder as the main document for isolated cases, or in a location that Xe_{La}TeX searches by default, e.g., `~/Library/texmf/xelatex/`. The package option `[no-config]` will suppress this behaviour under all circumstances.

★ v1.14: Used to be `[noconfig]`, which still works.

1.1.4 Warnings

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

1.2 About this manual

★ v1.6: An example warning!

In the unfortunate case that I need to make backwards incompatible changes (you're probably pretty safe these days), such things, and some other comments,

²<http://tug.ctan.org/cgi-bin/ctanPackageInformation.py?id=euenc>

³An example is distributed with the package.

are noted in the margin of this document as shown here, with a red star if the change is relevant to the current release of the package. (New features are denoted similarly in blue.)

This document has been typeset with X_YT_EX using a variety of fonts to display various features that the package supports. You will not be able to typeset the documentation if you do not have all of these fonts, many of which are distributed with Mac OS X or are otherwise commercial.

Many examples are shown in this manual. These are typeset side-by-side with their verbatim source code, although various size-altering commands (`\large`, `\Huge`, *etc.*) are omitted for clarity. Since the package supports font features for both AAT and OpenType fonts (whose feature sets only overlap to some extent), examples are distinguished by colour: blue and red, respectively. Examples whose font type is irrelevant are typeset in green.

2 Brief overview

This manual can get rather in-depth, as there are a lot of font features to cover. A basic preamble set-up is shown below, to simply select some default document fonts. See the file `fontspec-example.tex` for a more detailed example.

```
\usepackage{fontspec}
\defaultfontfeatures{Scale=MatchLowercase}
\setmainfont[Mapping=tex-text]{Baskerville}
\setsansfont[Mapping=tex-text]{Skia}
\setmonofont{Courier}
```

3 Font selection

`\fontspec` `\fontspec[]{}` is the base command of the package, used for selecting the specified ** in a L^AT_EX family. The font features argument accepts comma separated *=<option>* lists; these will not be fully described until Section 6 on page 14.

As our first example, look how easy it is to select the Hoefler Text typeface with the `fontspec` package:

The five boxing wizards jump quickly.
The five boxing wizards jump quickly.
THE FIVE BOXING WIZARDS JUMP QUICKLY.
THE FIVE BOXING WIZARDS JUMP QUICKLY.
The five boxing wizards jump quickly.
The five boxing wizards jump quickly.
THE FIVE BOXING WIZARDS JUMP QUICKLY.
THE FIVE BOXING WIZARDS JUMP QUICKLY.

```
\def\pangram{The five boxing
               wizards jump quickly.\}
\fontspec{Hoefler Text} \pangram
{\itshape \pangram}
{\scshape \pangram}
{\scshape\itshape \pangram}
\bfseries \pangram
{\itshape \pangram}
{\scshape \pangram}
{\itshape\scshape \pangram}
```

The fontspec package takes care of the necessary font definitions for those shapes as shown above *automatically*. Furthermore, it is not necessary to install the font for X_YTeX in any way whatsoever: every font that is installed in the operating system may be accessed.

3.1 Default font families

`\setmainfont` The `\setmainfont`,⁴ `\setsansfont`, and `\setmonofont` commands are used to select the default font families for the entire document. They take the same arguments as `\fontspec`. For example:

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

```
\setmainfont{Baskerville}
\setsansfont[Scale=0.86]{Skia}
\setmonofont[Scale=0.8]{Monaco}
\rmfamily\pangram\par
\sffamily\pangram\par
\ttfamily\pangram
```

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 5 on page 11, including methods for automatic scaling.

3.2 Font instances for efficiency

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

For this reason, *instances* of a font may be created with the `\newfontfamily` command, as shown in the following example:

★ v1.11: This macro used to be called `\newfontinstance`. Backwards compatibility is preserved via `fontspec.cfg`.

This is a *note*.

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

This macro should be used to create commands that would be used in the same way as `\rmfamily`, for example.

`\newfontface` Sometimes only a specific font face is desired, without accompanying italic or bold variants. This is common when selecting a fancy italic font, say, that has swash features unavailable in the upright forms. `\newfontface` is used for this purpose:

★ v1.11: New!

where is all the vegemite

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

This example is repeated in Section 6.6 on page 18.

⁴Or `\setromanfont`, a historical name that doesn't make much sense when you're, say, typesetting Greek.

3.3 Arbitrary bold/italic/small caps fonts

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

★ v1.6: These features used to be called `Bold` and `Italic`, and these shorter names may still be used if you desire.

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

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

★ v1.6: `BoldItalic` also works

For those cases that the base font name is repeated, you can replace it with an asterisk (first character only). For example, some space can be saved instead of writing 'Baskerville SemiBold':

Baskerville Italic SemiBold Italic	<code>\fontspec[BoldFont={* SemiBold}]{Baskerville}</code>
	<code>Baskerville \textit{Italic}</code>
	<code>\bfseries SemiBold \textit{Italic}</code>

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

Roman 123	<code>\fontspec[</code>
SMALL CAPS 456	<code>SmallCapsFont={Minion MM Small Caps & Oldstyle Figures},</code>
	<code>]{Minion MM Roman}</code>
	<code>Roman 123 \\\ \textsc{Small caps 456}</code>

3.4 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 L^AT_EX 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 you may find that loading some maths packages won't be as smooth as you expect since `fontspec` (and X_YL_AT_EX in general) breaks many of the assumptions of T_EX 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 (the latter loaded with `euler` rather than `eulervm`) should be fine; for all others keep an eye out for problems.

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

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

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

and this would allow you to typeset something like this:

$X \rightarrow X \rightarrow X$	<code>\$ X \rightarrow \mathrm{X} \rightarrow \mathbf{X} \$</code>
$X \rightarrow X \rightarrow X$	<code>\\boldmath</code>
$X \rightarrow X \rightarrow X$	<code>\$ X \rightarrow \mathrm{X} \rightarrow \mathbf{X} \$</code>

3.5 External fonts

X_YTeX v0.995 introduced the feature of loading fonts not installed through the operating system ('external' fonts). This feature is currently only available through the `xdvipdfmx` driver, which is notably *not* the default on Mac OS X.

This feature is handled in `fontspec` with the font feature `ExternalLocation`. When this feature is used, the main argument to `\fontspec` is the *file name* of the font (in contrast to the usual syntax which requires the font display name) and the argument to the feature is the (absolute) path to the font. For example:

```
\fontspec[ExternalLocation=/Users/will/Fonts/]{CODE2000.TTF}
```

If no path is given, then the font will be found in a location normally searched by X_YTeX, including the current directory. For example, the following declaration could load either the Latin Modern roman font in the current directory or, say, in `$TEXMF/fonts/opentype/public/lm/`:

```
\fontspec[ExternalLocation]{lmroman10-regular}
```

Bold and italic fonts cannot be automatically selected when external fonts are being used; they must be explicitly declared using the methods described in Section 3.3 on the preceding page.

3.6 Miscellaneous font selecting details

By the way, from v1.9, `\fontspec` and `\addfontfeatures` will now ignore following spaces as if it were a ‘naked’ control sequence; *e.g.*, ‘M. `\fontspec{...}` N’ and ‘M. `\fontspec{...}`N’ are the same.

Note that this package redefines the `\itshape` and `\scshape` commands in order to allow them to select italic small caps in conjunction. (This was implicitly shown in the first example, but it’s worth mentioning now, too.)

4 Selecting font features

The commands discussed so far each take an optional argument for accessing the font features of the requested font. These features are generally unavailable or harder to access in regular L^AT_EX. The font features and their options are described in Section 6 on page 14, but before we look at the range of available font features, it is necessary to discuss how they can be applied.

4.1 Default settings

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

Some ‘default’ Didot 0123456789 Now grey, with old-style figures: 0123456789	<pre>\fontspec{Didot} Some ‘default’ Didot 0123456789 \\\ \defaultfontfeatures{Numbers=OldStyle, Colour=888888} \fontspec{Didot} Now grey, with old-style figures: 0123456789</pre>
--	--

4.2 Changing the currently selected features

`\addfontfeatures` The `\addfontfeatures{}` command allows font features to be changed without knowing what features are currently selected or even what font is being used. A good example of this could be to add a hook to all tabular material to use monospaced numbers, as shown in the following example:

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

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

```
\fontspec[Numbers=OldStyle]{Skia}
`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 @{}}
\toprule Year & People & Miles & Boats \\\
\midrule 1842 & 999 & 75 & 13 \\\
          1923 & 111 & 54 & 56 \\\
\bottomrule
\end{tabular}}
```

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

4.3 Priority of feature selection

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

4.4 Different features for different font shapes

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.

ATTENTION ALL MARTINI DRINKERS
ATTENTION ALL MARTINI DRINKERS

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

Combined with the options for selecting arbitrary *fonts* for the different shapes, these separate feature options allow the selection of arbitrary weights in the Skia typeface, for example:

Skia
Skia 'Bold'

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

Note that because most fonts include their small caps glyphs within the main font, these features 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 following ludicrous example.

	<code>\fontspec[</code>
	<code>UprightFeatures={Colour = 220022,</code>
	<code>SmallCapsFeatures = {Colour=115511}},</code>
	<code>ItalicFeatures={Colour = 2244FF,</code>
	<code>SmallCapsFeatures = {Colour=112299}},</code>
	<code>BoldFeatures={Colour = FF4422,</code>
	<code>SmallCapsFeatures = {Colour=992211}},</code>
	<code>BoldItalicFeatures={Colour = 888844,</code>
	<code>SmallCapsFeatures = {Colour=444422}},</code>
	<code>] {Hoefer Text}</code>
Upright	<code>{\scshape Small Caps}\</code>
Italic	<code>{\itshape Italic {\scshape Italic Small Caps}\</code>
Bold	<code>{\bfseries Bold {\scshape Bold Small Caps}\</code>
Bold Italic	<code>{\itshape Bold Italic {\scshape Bold Italic Small Caps}}</code>

4.5 Different features for different font sizes

★ v1.13: New!

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.

Small	<code>\fontspec[SizeFeatures={</code>
Normal size	<code>{Size={-8}, Font=Apple Chancery, Colour=AA0000},</code>
Large	<code>{Size={8-14}, Colour=00AA00},</code>
	<code>{Size={14-}, Colour=0000AA}}] {Skia}</code>
	<code>{\scriptsize Small\par} Normal size\par {\Large Large\par}</code>

A less trivial example is shown in the context of optical font sizes in Section 6.2 on page 15.

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.

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

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

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

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

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

5 Font independent options

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

5.1 Scale

In its explicit form, `Scale` takes a single numeric argument for linearly scaling the font, as demonstrated in Section 3.1 on page 5. Since version 0.99 of \LaTeX , however, it is now possible to measure the correct dimensions of the fonts loaded, and hence calculate values to scale them automatically.

★ v1.9: As of Dec. 2005

The `Scale` feature now also takes the 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.

The perfect match is hard to find.
LOGO FONT

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

The amount of scaling used in each instance is reported in the `.log` file. Since there is some subjectivity about the exact scaling to be used, these values should be used to fine-tune the results.

5.2 Mapping

Mapping enables a \LaTeX text-mapping scheme.

“¡A small amount of—text!”

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

5.3 Colour

Colour (or Color), also shown in Section 4.1 on page 8 and Section 6 on page 14, uses X_YTeX font specifications to set the colour of the text. The colour is defined as a triplet of two-digit Hex RGB values, with optionally another value for the transparency (where 00 is completely transparent and FF is opaque.)



```
\fontsize{48}{48}
\fontspec{Hoefler Text Black}
{\addfontfeature{Color=FF000099}W}\kern-1ex
{\addfontfeature{Color=0000FF99}S}\kern-0.8ex
{\addfontfeature{Color=DDBB2299}P}\kern-0.8ex
{\addfontfeature{Color=00BB3399}R}
```

5.4 Interword space

While the space between words can be varied on an individual basis with the T_EX 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 by X_YTeX, and generally will not need to be adjusted. For those times when the precise details are important, the `WordSpace` features is provided, which takes either a single scaling factor to scale the value that X_YTeX has already chosen, or a triplet of comma-separated values for the nominal value, the stretch, and the shrink of the interword space, respectively. *I.e.*, `WordSpace=0.8` is the same as `WordSpace={0.8,0.8,0.8}`.

For example, I believe that the Cochin font, as distributed with Mac OS X, is too widely spaced. Now, this can be rectified, as shown below.

Some filler text for our example to take up some space, and to demonstrate the large default interword space in *Cochin*.

```
\fontspec{Cochin}
\fillertext
\space{1em}
```

Some filler text for our example to take up some space, and to demonstrate the large default interword space in *Cochin*.

```
\fontspec[ WordSpace = {0.7 , 0.8 , 0.9} ]{Cochin}
\fillertext
```

Be careful with the unpredictable things that the AAT font renderer can do with the text! Unlike T_EX, Mac OS X will allow fonts to letterspace themselves, which can be seen above; OpenType fonts, however, will not show this tendency, as they do not support this arguably dubious feature.

5.5 Post-punctuation space

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

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

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

```
\nonfrenchspacing
\fontspec{Baskerville}
Letters, Words. Sentences.      \par
\fontspec[PunctuationSpace=0.5]{Baskerville}
Letters, Words. Sentences.      \par
\fontspec[PunctuationSpace=0]{Baskerville}
Letters, Words. Sentences.
```

Also be aware that the above caveat for interword space also applies here, so after the last line in the above example, the `PunctuationSpace` for *all* Baskerville instances will be 0.

5.6 Letter spacing

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

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

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

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

This functionality *should not be used for lowercase text*, which is spacing correctly to begin with, but it can be very useful, in small amounts, when setting small caps or all caps titles. Also see the OpenType `Uppercase` option of the `Letters` feature (Section 6.4 on page 17).

5.7 The hyphenation character

The letter used for hyphenation may be chosen with the `HyphenChar` feature. It takes three types of input, which are chosen according to some simple rules. If

the input is the string `None`, then hyphenation is suppressed for this font. If the input is a single character, then this character is used. Finally, if the input is longer than a single character it must be the UTF-8 slot number of the hyphen character you desire.

Below, Adobe Garamond Pro’s uppercase hyphenation character⁵ is used to demonstrate a possible use for this feature. The second example redundantly demonstrates the default behaviour of using the hyphen as the hyphenation character.

<p>A MULTITUDE OF OBSTREPEROUSLY HYPHENATED ENTITIES</p> <p>A MULTITUDE OF OBSTREPER- OUSLY HYPHENATED ENTITIES</p> <p>A MULTITUDE OF OBSTREPER- OUSLY HYPHENATED ENTITIES</p>	<pre>\def\text {A MULTITUDE OF OBSTREPEROUSLY HYPHENATED ENTITIES \par\vspace{1ex}} \fontspec[HyphenChar=None]{Adobe Garamond Pro} \text \fontspec[HyphenChar={-}]{Adobe Garamond Pro} \text \fontspec[HyphenChar="F6BA]{Adobe Garamond Pro} \text</pre>
--	--

Note that in an actual situation, the Uppercase option of the Letters feature would probably supply this for you (see Section 6.4 on page 17).

The `xltextra` package redefines L^AT_EX’s `\-` macro such that it adjusts along with the above changes.

6 Font-dependent features

This section covers each and every font feature catered for by this package. Some, in fact, have already be seen in previous sections. There are too many to list in this introduction, but for a first taste of what is available, here is an example of the Apple Chancery typeface:

*My 1st example of
Apple Chancery*

```
\fontspec[
  Colour=CC00CC,
  Numbers=OldStyle,
  VerticalPosition=Ordinal,
  Variant=2]{Apple Chancery}
My 1st example of\ Apple Chancery
```

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

If a feature or an option is requested that the font does not have, a warning is given in the console output. As mentioned in 1.1.4 on page 3 these warnings can be suppressed by selecting the `[quiet]` package option.

⁵I found the character, and its number, in Mac OS X’s Character Palette.

6.1 Different font technologies: AAT and ICU

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

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

Among some other font features only available through a specific renderer, ICU provides for the `Script` and `Language` features, which allow different font behaviour for different alphabets and languages; see Section 6.19 on page 25 for the description of these features. *Because these font features can change which features are able to be selected for the font instance, they are selected by `fontspec` before all others and will automatically and without warning select the ICU renderer.*

6.2 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.

Optically sized fonts can be seen in either OpenType or Multiple Master varieties. The differences when dealing with these two are quite significant. OpenType fonts with optical scaling will exist in several discrete sizes, and these will be selected by X_YTeX *automatically* determined by the current font size. The `OpticalSize` option may be used to specify a different optical size.

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

	<code>\fontspec[OpticalSize=0]{Warnock Pro Caption}</code>
	Warnock Pro optical sizes
Warnock Pro optical sizes	<code>\fontspec[OpticalSize=0]{Warnock Pro}</code>
Warnock Pro optical sizes	Warnock Pro optical sizes
Warnock Pro optical sizes	<code>\fontspec[OpticalSize=0]{Warnock Pro Subhead}</code>
Warnock Pro optical sizes	Warnock Pro optical sizes
	<code>\fontspec[OpticalSize=0]{Warnock Pro Display}</code>
	Warnock Pro optical sizes

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

Automatic optical size	<code>\fontspec{Warnock Pro}</code>	
Automatic optical size	Automatic optical size	\\
Automatic optical size	<code>\scalebox{0.4}{\Huge</code>	
	Automatic optical size}	

Multiple Master fonts, on the other hand, are parameterised over orthogonal font axes, allowing continuous selection along such features as weight, width, and optical size (see Section 6.18 on page 25 for further details). Whereas an OpenType font will have only a few separate optical sizes, a Multiple Master font’s optical size can be specified over a continuous range. Unfortunately, this flexibility makes it harder to create an automatic interface through L^AT_EX, and the optical size for a Multiple Master font must always be specified explicitly.

	<code>\fontspec[OpticalSize=11]{Minion MM Roman}</code>	
MM optical size test	MM optical size test	\\
MM optical size test	<code>\fontspec[OpticalSize=47]{Minion MM Roman}</code>	
MM optical size test	MM optical size test	\\
	<code>\fontspec[OpticalSize=71]{Minion MM Roman}</code>	
	MM optical size test	\\

The `SizeFeatures` feature (Section 4.5 on page 10) can be used to specify exactly which optical sizes will be used for ranges of font size. For example, something like


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

6.3 Ligatures

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

The first three are also supported in OpenType fonts, which may also use `Historical` and `Contextual`. To turn a ligature option *off*, prefix its name with `No`: *e.g.*, `NoDiphthong`.

strict firefly	<code>\fontspec[Ligatures=Rare]{Hoefler Text}</code>	
strict firefly	strict firefly	\\
strict firefly	<code>\fontspec[Ligatures=NoCommon]{Hoefler Text}</code>	
	strict firefly	

Rare: Ð Þ ð þ
 Logos: 
 Rebus: %
 Diphthong: Æ Œ æ œ

```
\fontspec
  [Ligatures={Rare,Logos,Rebus,Diphthong}]
  {Palatino}
Rare: Dh Th dh th      \\
Logos: apple           \\
Rebus: \%0             \\
Dipht\null hong: AE OE ae oe
```

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

6.4 Letters

★ v1.6: This feature has changed names along with its options, **breaking** backwards compatibility!

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

THIS SENTENCE NO VERB
 this sentence no verb
 This Sentence No Verb

```
\fontspec[Letters=Uppercase]{Palatino}
THIS Sentence no verb      \\
\fontspec[Letters=Lowercase]{Palatino}
THIS Sentence no verb      \\
\fontspec[Letters=InitialCaps]{Palatino}
THIS Sentence no verb
```

★ v1.9: The `Uppercase`... variants have changed (e.g., from `SMALLCAPS`) to allow for more flexible option handling in the future. The old forms still work, for now...

OpenType fonts have some different options: `Uppercase`, `SmallCaps`, `PetiteCaps`, `UppercaseSmallCaps`, `UppercasePetiteCaps`, and `Unicase`. Petite caps are smaller than small caps. Mixed case commands turn lowercase letters into the smaller caps letters, whereas uppercase options turn the capital letters to the smaller caps (good, e.g., for applying to already uppercase acronyms like ‘NASA’). ‘Unicase’ is a weird hybrid of upper and lower case letters.

THIS SENTENCE NO VERB
 THIS SENTENCE no verb

```
\fontspec[Letters=SmallCaps]{Warnock Pro}
THIS SENTENCE no verb      \\
\fontspec[Letters=UppercaseSmallCaps]{Warnock Pro}
THIS SENTENCE no verb
```

The `Uppercase` option is also provided *but* it will (probably) not actually map letters to uppercase.⁶ It will, however, select various uppercase forms for glyphs such as accents and dashes.

UPPER-CASE EXAMPLE
 UPPER-CASE EXAMPLE

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

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

The Kerning feature also contains an Uppercase option, which adds a small amount of spacing in between letters (see Section 6.13 on page 22). This feature was originally planned to be included with the one above (so Letters=Uppercase would do both punctuation *and* tracking), but I decided that it would be a bad idea to break the one-to-one correspondence with fontspec and OpenType features. (Sorry TUGboat readers!)

6.5 Numbers

The Numbers feature defines how numbers will look in the selected font. For both AAT and OpenType 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 4.2 on page 8.

For OpenType fonts, there is also the SlashedZero option which replaces the default zero with a slashed version to prevent confusion with an uppercase ‘O’.

0123456789 0123456789	<pre>\fontspec[Numbers=Lining]{Warnock Pro} 0123456789 \fontspec[Numbers=SlashedZero]{Warnock Pro} 0123456789</pre>
-----------------------	---

6.6 Contextuals

★ v1.9: This feature used to be called Swashes. This name still works, for now.

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

<i>where is all the vegemite</i>	<pre>\newfontface\fancy [Contextuals={WordInitial,WordFinal}] {Hoefer Text Italic} \fontspec{fancy} where is all the vegemite</pre>
‘Inner’ fwafhes can <i>fometimes</i> contain the archaic long s.	<pre>\fontspec[Contextuals=Inner]{Hoefer Text} ‘Inner’ swafhes can \emph{sometimes} contain the archaic long~s.</pre>

For OpenType fonts, all features as above but the LineInitial feature are supported, and Swash turns on contextual swashes.

★ v1.9: Used to be Contextual; still works.

<i>Without Contextual Swashes</i> <i>With Contextual Swashes; cf. W C S</i>	<pre>\fontspec{Warnock Pro} \itshape Without Contextual Swashes \fontspec[Contextuals=Swash]{Warnock Pro} With Contextual Swashes; cf. W C S</pre>
--	--

Historic forms (e.g., long s as shown above) are accessed in OpenType fonts via the feature `Style=Historic`; this is generally *not* contextual in OpenType, which is why it is not included here.

6.7 Vertical position

The `VerticalPosition` feature is used to access things like subscript (`Superior`) and superscript (`Inferior`) 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.

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

OpenType fonts also have the option `ScientificInferior` which extends further below the baseline than `Inferiors`, as well as `Numerator` and `Denominator` for creating arbitrary fractions (see next section). Beware, the `Ordinal` feature will not work correctly for all OpenType fonts!

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

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

6.8 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.

* v1.7: This feature has changed:
no backwards compatibility!

In AAT fonts, the 'fraction slash' or solidus character, which may be obtained by typing `'\c 1'`, is (supposed) to be used to create fractions. When `Fractions` are turned `On`, then (supposedly) only pre-drawn fractions will be used.

½ 5/6
1/2 5/6

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

Using the Diagonal option (AAT only), the font will attempt to create the fraction from superscript and subscript characters. This is shown in the following example by Hoefler Text, whose fraction support may actually not be turned off.

13579/24680
13579/24680

```
\fontspec{Hoefler Text}
13579 24680 \\ % fraction slash
\quad 13579/24680 % regular slash
```

OpenType fonts simply use a regular text slash to create fractions:

1/2 1/4 5/6 13579/24680
½ ¼ ⅝ 13579/24680

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

Some (Asian fonts predominantly) also provide for the Alternate feature:

1/2 1/4 5/6 13579/24680
½ ¼ ⅝ 13579/24680

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

The xltextra package provides a \vfrac command for creating arbitrary so-called ‘vulgar’ fractions:

13579/24680

```
\fontspec{Warnock Pro}
\vfrac{13579}{24680}
```

6.9 Variants

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



```
\newcounter{var}\newcounter{trans}
\whiledo{\value{var}<9}{%
\stepcounter{trans}%
\fontspec[Variant=\thevar,
Colour=005599\thetrans\thetrans]{Zapfino}%
\makebox[0.75\width]{d}%
\stepcounter{var}}
```

For OpenType fonts, Variant selects a ‘Stylistic Set’, again specified numerically. I don’t have a font to demonstrate this feature with, unfortunately. See Section 7 on page 27 for a way to assign names to variants, which should be done on a per-font basis.

6.10 AAT Alternates

Selection of Alternates in AAT fonts *again* must be done numerically.

<i>Sphinx Of Black Quartz, JUDGE Mr Vow</i>	<code>\fontspec[Alternate=0]{Hoefler Text Italic}</code>
<i>Sphinx Of Black Quartz, JUDGE Mr</i>	<code>Sphinx Of Black Quartz, {\scshape Judge My Vow} \</code>
<i>Vow</i>	<code>\fontspec[Alternate=1]{Hoefler Text Italic}</code>
	<code>Sphinx Of Black Quartz, {\scshape Judge My Vow}</code>

See Section 7 on page 27 for a way to assign names to alternates, which should be done on a per-font basis.

6.11 Style

★ v1.7: The old name, `StyleOptions`, still works.

The options of the Style feature are defined in AAT as one of the following: Display, Engraved, IlluminatedCaps, Italic, Ruby,⁷ TallCaps, or TitlingCaps.

<i>[ABCD...WXYZ]</i>	<code>\newfontface\officedoor[Style=Engraved]{Hoefler Text}</code>
	<code>\officedoor [ABCD\dots WXYZ]</code>

ICU supported options are Alternate, Italic, Historic, Ruby,⁷ Swash, TitlingCaps, HorizontalKana, and VerticalKana.

<i>KQRkvw y</i>	<code>\fontspec{Warnock Pro}</code>
<i>KQRkvw y</i>	<code>K Q R k v w y \</code>
	<code>\addfontfeature{Style=Alternate}</code>
	<code>K Q R k v w y</code>

Note the occasional inconsistency with which font features are labelled; a long-tailed 'Q' could turn up anywhere!

<i>M Q Z</i>	<code>\fontspec{Adobe Jenson Pro}</code>
<i>M Q Z</i>	<code>M Q Z \</code>
	<code>\addfontfeature{Style=Historic}</code>
	<code>M Q Z</code>

<i>TITLING CAPS</i>	<code>\fontspec{Adobe Garamond Pro}</code>
<i>TITLING CAPS</i>	<code>TITLING CAPS \</code>
	<code>\addfontfeature{Style=TitlingCaps}</code>
	<code>TITLING CAPS</code>

Two features in one example; Italic affects the Latin text and Ruby the Japanese:

<i>Latin ようこそ ワカヨタレソ</i>	<code>\fontspec{Hiragino Mincho Pro W3}</code>
<i>Latin ようこそ ワカヨタレソ</i>	<code>Latin ようこそ ワカヨタレソ \</code>
	<code>\addfontfeature{Style={Italic, Ruby}}</code>
	<code>Latin ようこそ ワカヨタレソ</code>

Note the difference here between the default and the horizontal style kana:

⁷'Ruby' refers to a small optical size, used in Japanese typography for annotations.

ようこそ ワカヨタレソ
 ようこそ ワカヨタレソ
 ようこそ ワカヨタレソ

```
\fontspec{Hiragino Mincho Pro}
ようこそ ワカヨタレソ \\
{\addfontfeature{Style=HorizontalKana}
ようこそ ワカヨタレソ} \\
{\addfontfeature{Style=VerticalKana}
ようこそ ワカヨタレソ}
```

6.12 Diacritics

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

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

Ó Ö Ø
 O' O" O/
 Better: Ó Ö Ø

```
\fontspec[Diacritics=Show]{Palatino}
O' \quad O" \quad O/ \par
\fontspec[Diacritics=Decompose]{Palatino}
O' \quad O" \quad O/ \par
Better: \'0 \"0 \% (requires xunicode)
```

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

No options for OpenType fonts.

6.13 Kerning

Well designed fonts contain kerning information that controls the spacing between letter pairs, on an individual basis. The Kerning feature provides options to control this, for OpenType fonts only.

The options provided for now are On, Off (don't know why you'd want to), and Uppercase.

Ta AV
 Ta AV

```
\fontspec{Warnock Pro}
Ta AV \\
\fontspec[Kerning=Off]{Warnock Pro}
Ta AV
```

As briefly mentioned previously at the end of Section 6.4 on page 17, the Uppercase option will add a small amount of tracking between uppercase letters:

UPPER-CASE EXAMPLE
 UPPER-CASE EXAMPLE

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

6.14 CJK shape

★ v1.9: Was `CharacterShape`, which wasn't very descriptive. No backwards compatibility.

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.

㗎㗎㗎 妍并訝
㗎㗎㗎 妍并訝
㗎㗎㗎 妍并訝

```
\fontspec{Hiragino Mincho Pro}
{\addfontfeature{CJKShape=Traditional}
  㗎㗎㗎 妍并訝 }
{\addfontfeature{CJKShape=NLC}
  㗎㗎㗎 妍并訝 }
{\addfontfeature{CJKShape=Expert}
  㗎㗎㗎 妍并訝 }
```

6.15 Character width

★ v1.9: Was `TextSpacing`, which wasn't very descriptive. No backwards compatibility.

Many Asian fonts are equipped with variously spaced characters for shoe-horning into their generally monospaced text. These are accessed through the `CharacterWidth` feature.⁸ For now, OpenType and AAT share the same six options for this feature: `Proportional`, `Full`, `Half`, `Third`, `Quarter`, `AlternateProportional`, and `AlternateHalf`. AAT also allows `Default` to return to whatever was originally specified.

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.

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

⁸Apple seems to be adapting its AAT features in this regard (at least in the fonts it distributes with Mac OS X) to have a one-to-one correspondence with the equivalent OpenType features. Previously AAT was more fine grained, but naturally they're not documenting their AAT tables any more, so if the following features don't work for a specific font let me know and I'll try and see if anything can be salvaged from the situation.

The same situation occurs with numbers, which are provided in increasingly illegible compressed forms:

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

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

6.16 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`.

<p>1 2 3 4 5 6 7 8 9 ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ (1) (2) (3) (4) (5) (6) (7) (8) (9) 1. 2. 3. 4. 5. 6. 7. 8. 9.</p>	<pre>\fontspec{Hei Regular} 1 2 3 4 5 6 7 8 9 \fontspec[Annotation=Circle]{Hei Regular} 1 2 3 4 5 6 7 8 9 \fontspec[Annotation=Parenthesis]{Hei Regular} 1 2 3 4 5 6 7 8 9 \fontspec[Annotation=Period]{Hei Regular} 1 2 3 4 5 6 7 8 9</pre>
--	--

For OpenType fonts, the only option supported is `On` and `Off`:

<p>1 2 3 4 5 6 7 8 9 (1) (2) (3) (4) (5) (6) (7) (8) (9)</p>	<pre>\fontspec{Hiragino Maru Gothic Pro} 1 2 3 4 5 6 7 8 9 \addfontfeature{Annotation=On} 1 2 3 4 5 6 7 8 9</pre>
---	---

I'm not sure if \XeTeX can access alternate annotation forms, even if they exist (as in this case) in the font.

6.17 Vertical typesetting

A recent feature of \XeTeX is the ability to rotate the glyphs in AAT fonts by 90° , providing a method to typeset vertically by building a horizontal box as normal and then rotating it.

共產主義者は

共
産
主
義
者
は

```
\fontspec{Hiragino Mincho Pro}  
共產主義者は
```

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

The AAT renderer is required above because X_YTeX choses the ICU renderer by preference when both options are available; if it is not explicitly chosen, the glyphs will not be rotated and a warning will be printed in the output.

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

6.18 AAT & Multiple Master font axes

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

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

Really light and extended Skia
Really fat and condensed Skia

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

Variations along a multiple master font's optical size axis has been shown previously in Section 6.2 on page 15.

6.19 OpenType scripts and languages

When dealing with fonts that include glyphs for various languages, they may contain different font features for the different character sets and languages it supports. These may be selected with the Script and Language features. The possible options are tabulated in Table 2 on page 27 and Table 3 on page 28, respectively. When a script or language is requested that is not supported by the current font, a warning is printed in the console output.

Because these font features can change which features are able to be selected for the font, they are selected by fontspec before all others and will specifically select the ICU renderer for this font, as described in Section 6.1 on page 15.

6.19.1 Script examples

In the following examples, the same font is used to typeset the verbatim input and the X_YTeX output. Because the `Script` is only specified for the output, the text is rendered incorrectly in the verbatim input. 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.

العربي	<code>\fontspec[Script=Arabic]{Code2000}</code> العر بي
हिन्दी	<code>\fontspec[Script=Devanagari]{Code2000}</code> हन्दि
লেখ	<code>\fontspec[Script=Bengali]{Code2000}</code> লৎ
મયાદા-સૂચક નિવેદન	<code>\fontspec[Script=Gujarati]{Code2000}</code> મયાદા-સૂચક નિવેદન
നമുടെ പാരമ്പര്യ	<code>\fontspec[Script=Malayalam]{Code2000}</code> നമുടേ പാരമ്പര്യ
ਆਦਿ ਸਚੁ ਜੁਗਾਦਿ ਸਚੁ	<code>\fontspec[Script=Gurmukhi]{Code2000}</code> ਆਦਿ ਸਚੁ ਜੁਗਾਦਿ ਸਚੁ
தமிழ் தேடி	<code>\fontspec[Script=Tamil]{Code2000}</code> தமிழ் தேடி
תּוֹרָה	<code>\fontspec[Script=Hebrew]{Code2000}</code> תּוֹרָה

6.19.2 Language examples

Vietnamese requires careful diacritic placement:

cấp số mỗĩ	<code>\fontspec{Doulos SIL}</code>
cấp số mỗĩ	<code>cấp số mỗĩ \\</code>
	<code>\addfontfeature{Language=Vietnamese}</code>
	<code>cấp số mỗĩ</code>

Moldavian, as a typical example from Ralf Stubner's FPL Neu font:

Ș ș Ţ ţ	<code>\fontspec{FPL Neu}</code>
Ș ș Ţ ţ	<code>Ș ș Ţ ţ \\</code>
	<code>\addfontfeature{Language=Moldavian}</code>
	<code>Ș ș Ţ ţ</code>

6.19.3 Defining new scripts and languages

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

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

The first argument is the fontspec name, the second the OpenType definition. The advantage to using these commands rather than `\newfontfeature` (see Section 7) is the error-checking that is performed when the script or language is requested.

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

Table 2: Defined Scripts for OpenType fonts. Aliased names are shown in adjacent positions marked with red pilcrows (☞), defined in `fontspec.cfg`.

7 Defining new features

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

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

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

Use the X_YTeX file `AAT-info.tex` to obtain the code numbers. For example:

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

This command replaces `\newfontfeaturecode`, which is provided for backwards compatibility via `fontspec.cfg`.

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

Table 3: Defined Languages for OpenType fonts. Note that they are sorted alphabetically *not* by name but by OpenType tag, which is a little irritating, really.

`\newICUfeature`

New OpenType features may be created with this command:

`\newICUfeature{<feature>}{<option>}{<feature tag>}`

In the following example, the Moldavian language (see Section 6.19 on page 25) must be activated to achieve the effect shown.

Ș ș Ț ț
Ș ș Ț ț

```
\newICUfeature{Style}{NoLocalForms}{-loc1}
\fontspec[Language=Moldavian]{FPL Neu}
Ș ș Ț ț \
\addfontfeature{Style=NoLocalForms}
Ș ș Ț ț
```

`\newfontfeature`

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

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

For example, Zapfino contains the feature 'Avoid d-collisions'. To access it with this package, you could do the following:

sockdolager rubdown
sockdolager rubdown

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

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

7.1 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:

Roman Letters *And Swash*

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

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{}{<existing name>}{<new name>}`:

Scientific
Inferior: 12345

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

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

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

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

7.2 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^AT_EX font selection conveniences. The `RawFeaturefont` feature allows literal X_YT_EX font feature selection when you happen to have the OpenType feature tag memorised.

★ v1.13: New!

FPL NEU SMALL CAPS

```
\fontspec[RawFeature=+smcp]{FPL Neu}
FPL Neu small caps
```

Multiple features can either be included in a single declaration:

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

or with multiple declarations:

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

File I

fontspec.sty

8 Implementation

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

For some reason, I decided to prefix all the package internal command names and variables with `zf`. I don't know why I chose those letters, but I guess I just liked the look/feel of them together at the time. (Possibly inspired by Hermann Zapf.)

Only proceed if it is \LaTeX that is doing the typesetting.

```
1 \RequirePackage{ifxetex}
2 \RequireXeTeX
```

8.1 Bits and pieces

Conditionals

```
3 \newif\ifzf@firsttime
4 \newif\ifzf@nobf
5 \newif\ifzf@noit
6 \newif\ifzf@nosc
7 \newif\ifzf@tfm
8 \newif\ifzf@atsui
9 \newif\ifzf@icu
10 \newif\ifzf@mm
```

For dealing with legacy maths

```
11 \newif\ifzf@math@euler
12 \newif\ifzf@math@lucida
13 \newif\ifzf@package@euler@loaded
```

For, well, dealing with babel:

```
14 \newif\ifzf@package@babel@loaded
```

For package options:

```
15 \newif\if@zf@configfile
16 \newif\if@zf@euenc
17 \newif\if@zf@math
```

Counters

```
18 \newcount\c@zf@newff
19 \newcount\c@zf@index
20 \newcount\c@zf@script
21 \newcount\c@zf@language
```


fontspec shorthands:

```
22 \newcommand\zf@PackageError[2]{\PackageError{fontspec}{#1}{#2}}
23 \newcommand\zf@PackageWarning[1]{\PackageWarning{fontspec}{#1}}
24 \newcommand\zf@PackageInfo[1]{\PackageInfo{fontspec}{#1}}
```

\def@cx L^AT_EX₃-like syntax for various low level commands. Makes life much easier; can't wait for the official interface :)

```
\gdef@cx
\let@cc 25 \providecommand\def@cx[2]{\expandafter\edef\csname#1\endcsname{#2}}
26 \providecommand\gdef@cx[2]{\expandafter\xdef\csname#1\endcsname{#2}}
27 \providecommand\let@cc[2]{%
28 \expandafter\let\csname#1\expandafter\endcsname\csname#2\endcsname}
```

8.2 Option processing

```
29 \DeclareOption{cm-default}{\@zf@euencfalse}
30 \DeclareOption{lm-default}{\@zf@euenctrue}
31 \DeclareOption{math}{\@zf@mathtrue}
32 \DeclareOption{no-math}{\@zf@mathfalse}
33 \DeclareOption{config}{\@zf@configfiletrue}
34 \DeclareOption{no-config}{\@zf@configfilefalse}
35 \DeclareOption{noconfig}{\@zf@configfilefalse}
36 \DeclareOption{quiet}{\let\zf@PackageWarning\zf@PackageInfo}
37 \ExecuteOptions{config,lm-default,math}
38 \ProcessOptions*
```

Only proceed if it is X_YT_EX that is doing the typesetting:

```
39 \RequirePackage{ifxetex}
40 \RequireXeTeX
```

8.3 Packages

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

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

8.4 Encodings

Frank Mittelbach has recommended using the ‘EUx’ family of font encodings to experiment with unicode. Now that X_YT_EX can find fonts in the texmf tree, the Latin Modern OpenType fonts can be used as the defaults. See the euenc collection of files for how this is implemented.

```
43 \if@zf@euenc
44 \def\zf@enc{EU1}
45 \renewcommand{\rmdefault}{lmr}
46 \renewcommand{\sfdefault}{lms}
47 \renewcommand{\ttdefault}{lmtt}
```

```

48 \RequirePackage[\zf@enc]{fontenc}
49 \else
50 \def\zf@enc{U}
51 \let\encodingdefault\zf@enc
52 \fi
53 \let\UTFencname\zf@enc

```

Dealing with a couple of the problems introduced by babel:

```

54 \let\cyrillicencoding\zf@enc
55 \let\latinencoding\zf@enc
56 \g@addto@macro\document{%
57 \let\cyrillicencoding\zf@enc
58 \let\latinencoding\zf@enc}

```

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

8.5 User commands

This section contains the definitions of the commands detailed in the user documentation. Only the ‘top level’ definitions of the commands are contained herein; they all use or define macros which are defined or used later on in Section 8.6 on page 37.

8.5.1 Font selection

`\fontspec` This is the main command of the package that selects fonts with various features. It takes two arguments: the Mac OS X font name and the optional requested features of that font. It simply runs `\zf@fontspec`, which takes the same arguments as the top level macro and puts the new-fangled font family name into the global `\zf@family`. Then this new font family is selected.

```

59 \newcommand*\fontspec[2][]{%
60 \zf@fontspec{#1}{#2}%
61 \fontfamily\zf@family\selectfont
62 \ignorespaces}

```

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

```

63 \newcommand*\setmainfont[2][]{%
64 \zf@fontspec{#1}{#2}%
65 \let\rmdefault\zf@family
66 \normalfont}

```

```

67 \let\setromanfont\setmainfont
68 \newcommand*\setsansfont[2][]{%
69   \zf@fontspec{#1}{#2}%
70   \let\sfddefault\zf@family
71   \normalfont}
72 \newcommand*\setmonofont[2][]{%
73   \zf@fontspec{#1}{#2}%
74   \let\ttdefault\zf@family
75   \normalfont}

```

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

```

76 \newcommand*\setmathrm[2][]{%
77   \zf@fontspec{#1}{#2}%
78   \let\zf@rmmaths\zf@family}
79 \newcommand*\setboldmathrm[2][]{%
80   \zf@fontspec{#1}{#2}%
81   \let\zf@rmboldmaths\zf@family}
82 \newcommand*\setmathsf[2][]{%
83   \zf@fontspec{#1}{#2}%
84   \let\zf@sfmaths\zf@family}
85 \newcommand*\setmathtt[2][]{%
86   \zf@fontspec{#1}{#2}%
87   \let\zf@ttmaths\zf@family}
88 \@onlypreamble\setmathrm
89 \@onlypreamble\setboldmathrm
90 \@onlypreamble\setmathsf
91 \@onlypreamble\setmathtt

```

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

```

92 \def\zf@rmmaths{\rmdefault}
93 \def\zf@sfmaths{\sfdefault}
94 \def\zf@ttmaths{\ttdefault}

```

`\newfontfamily` This macro takes the arguments of `\fontspec` with a prepended *(instance cmd)*
`\newfontface` (code for middle optional argument generated by Scott Pakin's `newcommand.py`).
 This command is used when a specific font instance needs to be referred to repet-
 itively (*e.g.*, in a section heading) since continuously calling `\zf@fontspec` is inef-
 ficient because it must parse the option arguments every time.

`\zf@fontspec` defines a font family and saves its name in `\zf@family`. This
 family is then used in a typical NFSS `\fontfamily` declaration, saved in the macro
 name specified.

```

95 \newcommand*\newfontfamily[1]{%
96   \@ifnextchar[{\newfontfamily@i#1}{\newfontfamily@i#1[]}}

```

```

97 \def\newfontfamily@i#1[#2]#3{%
98   \zf@fontspec{#2}{#3}%
99   \edef\@tempa{%
100     \noexpand\DeclareRobustCommand\noexpand#1
101     {\noexpand\fontfamily{\zf@family}\noexpand\selectfont}}%
102   \@tempa}

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

103 \newcommand*\newfontface[1]{%
104   \@ifnextchar[{\newfontface@i#1}{\newfontface@i#1[]}}
105 \def\newfontface@i#1[#2]#3{%
106   \zf@fontspec{BoldFont={},ItalicFont={},SmallCapsFont={},#2}{#3}%
107   \edef\@tempa{%
108     \noexpand\DeclareRobustCommand\noexpand#1
109     {\noexpand\fontfamily{\zf@family}\noexpand\selectfont}}%
110   \@tempa}

```

8.5.2 Font feature selection

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

```

111 \newcommand*\defaultfontfeatures[1]{\def\zf@default@options{#1,}}
112 \let\zf@default@options\@empty

```

`\addfontfeatures` In order to be able to extend the feature selection of a given font, two things need to be known: the currently selected features, and the currently selected font. Every time a font family is created, this information is saved inside a control sequence with the name of the font family itself.

This macro extracts this information, then appends the requested font features to add to the already existing ones, and calls the font again with the top level `\fontspec` command.

The default options are *not* applied (which is why `\zf@default@options` is emptied inside the group; this is allowed as `\zf@family` is globally defined in `\zf@fontspec`), so this means that the only added features to the font are strictly those specified by this command.

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

```

113 \newcommand*\addfontfeatures[1]{%
114   \begingroup
115     \let\zf@default@options\@empty
116     \edef\@tempa{%
117       \noexpand\zf@fontspec

```

```

118      {\csname zf@family@options\f@family\endcsname,#1}%
119      {\csname zf@family@fontname\f@family\endcsname}}}%
120    \@tempa
121  \endgroup
122  \fontfamily\zf@family\selectfont
123  \ignorespaces}
124  \let\addfontfeature\addfontfeatures

```

8.5.3 Defining new font features

`\newfontfeature` `\newfontfeature` takes two arguments: the name of the feature tag by which to reference it, and the string that is used to select the font feature. It uses a counter to keep track of the number of new features introduced; every time a new feature is defined, a control sequence is defined made up of the concatenation of `+zf-` and the new feature tag. This long-winded control sequence is then called upon to update the font family string when a new instance is requested.

```

125 \newcommand*\newfontfeature[2]{%
126   \stepcounter{zf@newff}%
127   \def@cx{+zf-#1}{+zf-\the\c@zf@newff}%
128   \define@key{zf}{options}{#1}[]{}%
129   \zf@update@family{\csname+zf-#1\endcsname}%
130   \zf@update@ff{#2}}

```

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

```

131 \newcommand*\newAATfeature[4]{%
132   \unless\ifcsname zf@options@#1\endcsname
133     \zf@define@font@feature{#1}%
134   \fi
135   \key@ifundefined{zf}{#1}{#2}{}{}%
136   \zf@PackageWarning{Option '#2' of font feature '#1' overwritten.}}%
137   \zf@define@feature@option{#1}{#2}{#3}{#4}{}

```

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

```

138 \newcommand*\newICUfeature[3]{%
139   \unless\ifcsname zf@options@#1\endcsname
140     \zf@define@font@feature{#1}%
141   \fi
142   \key@ifundefined{zf}{#1}{#2}{}{}%
143   \zf@PackageWarning{Option '#2' of font feature '#1' overwritten.}}%
144   \zf@define@feature@option{#1}{#2}{}{}{#3}

```

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

`\aliasfontfeatureoption` I've been consistent, they should work for everything.

```

145 \newcommand*\aliasfontfeature[2]{\multi@alias@key{#1}{#2}}
146 \newcommand*\aliasfontfeatureoption[3]{\keyval@alias@key[zf@feat]{#1}{#2}{#3}}

```

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

```

147 \newcommand*\newfontscript[2]{%
148   \define@key[zf@feat]{Script}{#1}[]{}%
149   \zf@check@ot@script{#2}%
150   \if@tempswa
151     \global\c@zf@script\@tempcnta\relax
152     \xdef\zf@script@name{#1}%
153     \xdef\zf@family@long{\zf@family@long+script=#1}%
154     \xdef\zf@pre@ff{script=#2,\zf@pre@ff}%
155   \else
156     \zf@PackageWarning{Font \fontname\zf@basefont does not con-
      tain script '#1'}%
157   \fi}}

```

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

```

158 \newcommand*\newfontlanguage[2]{%
159   \define@key[zf@feat]{Lang}{#1}[]{}%
160   \zf@check@ot@lang{#2}%
161   \if@tempswa
162     \global\c@zf@language\@tempcnta\relax
163     \xdef\zf@language@name{#1}%
164     \xdef\zf@family@long{\zf@family@long+lang=#1}%
165     \xdef\zf@pre@ff{\zf@pre@ff language=#2,}%
166   \else
167     \zf@PackageWarning{%
168       Font \fontname\zf@basefont does not contain
169       language '#1' for script '\zf@script@name'}%
170   \fi}}

```

8.6 Internal macros

`\zf@fontspec` This is the command that defines font families for use, the underlying procedure of all `\fontspec`-like commands. Given a list of font features (#1) for a requested

font (#2, stored in \zf@fontname globally for the \zf@make@aat@feature@string macro), it will define an NFSS family for that font and put the family name into \zf@family.

This macro does its processing inside a group, but it's a bit worthless coz there's all sorts of \global action going on. Pity. Anyway, lots of things are branched out for the pure reason of splitting the code up into logical chunks. Some of it is never even re-used, so it all might be a bit obfuscating. (E.g., \zf@init and \zf@set@font@type.)

First off, initialise some bits and pieces and run the preparse feature processing. This catches font features such as Renderer that can change the way subsequent features are processed. All font features that 'slip through' this stage are saved in the \zf@font@feat macro for future processing.

```

171 \newcommand*\zf@fontspec[2]{%
172   \begingroup
173   \zf@init
174   \edef\zf@fontname{#2}%
175   \let\zf@family@long\zf@fontname
176   \setkeys*[zf]{preparse}{#1}%
177   \edef\@tempa{\noexpand\setkeys*[zf]{preparse}{\XKV@rm}}\@tempa
178   \let\zf@font@feat\XKV@rm

```

Now check if the font is to be rendered with ATSUI or ICU. This will either be automatic (based on the font type), or specified by the user via a font feature. If automatic, the \zf@suffix macro will still be empty (other suffices that could be added will be later in the feature processing), and if it is indeed still empty, assign it a value so that the other weights of the font are specifically loaded with the same renderer. This fixes a bug in v1.10 for a mishmash of Lucida fonts.

```

179   \font\zf@basefont="\zf@font@wrap\zf@fontname\zf@suffix" at \f@size pt
180   \unless\ifzf@icu
181     \zf@set@font@type
182     \fi
183     \ifx\zf@suffix\@empty
184       \ifzf@atsui
185         \def\zf@suffix{/AAT}%
186       \else
187         \ifzf@icu
188           \def\zf@suffix{/ICU}%
189         \fi
190       \fi
191       \font\zf@basefont="\zf@font@wrap\zf@fontname\zf@suffix" at \f@size pt
192     \fi

```

Now convert the remaining requested features to font definition strings. This is performed with \zf@get@feature@requests, in which \setkeys retrieves the requested font features and processes them. To build up the complex family name, it concatenates each font feature with the family name of the font. So since

\setkeys is run more than once (since different font faces may have different feature names), we only want the complex family name to be built up once, hence the \zf@firsttime conditionals.

In the future, this will be replaced by a dedicated `makefamilyxkeyval\setkeys` declaration. Probably.

```
193 \zf@firsttimetrue
194 \zf@get@feature@requests{\zf@font@feat}%
195 \zf@firsttimefalse
```

Now we have a unique (in fact, too unique!) string that contains the family name and every option in abbreviated form. This is used with a counter to create a simple NFSS family name for the font we're selecting.

```
196 \unless\ifcsname zf@UID@\zf@family@long\endcsname
197 \ifcsname c@zf@famc@#2\endcsname
198 \expandafter\stepcounter\else
199 \expandafter\newcounter\fi
200 {zf@famc@#2}%
201 \def@cx{zf@UID@\zf@family@long}{%
202 \zap@space#2 \@empty
203 (\expandafter\the\csname c@zf@famc@#2\endcsname)}%
204 \fi
205 \xdef\zf@family{\@nameuse{zf@UID@\zf@family@long}}%
```

Now that we have the family name, we can check to see if the family has already been defined, and if not, do so. Once the family name is created, use it to create global macros to save the user string of the requested options and font name, primarily for use with `\addfontfeatures`.

```
206 \unless\ifcsname zf@family@fontname\zf@family\endcsname
207 \zf@PackageInfo{Defining font family for '#2'
208 with options [\zf@default@options #1]}%
209 \gdef@cx{zf@family@fontname\zf@family}{\zf@fontname}%
210 \gdef@cx{zf@family@options\zf@family}{\zf@default@options #1}%
211 \gdef@cx{zf@family@fontdef\zf@family}
212 {\zf@fontname\zf@suffix:\zf@pre@ff\zf@ff}%
```

Next the font family and its shapes are defined in the NFSS.

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

The macros `\zf@bf`, et al., are used to store the name of the custom bold, et al., font, if requested as user options. If they are empty, the default fonts are used.

First we define the font family and define the normal shape: (any shape-specific features are appended to the generic font features requested in the last argument of `\zf@make@font@shapes`.)

```
213 \DeclareFontFamily{\zf@enc}{\zf@family}{}%
214 \zf@make@font@shapes{\zf@fontname}
```



```
215      {\mddefault}{\updefault}{\zf@font@feat\zf@up@feat}%
```

Secondly, bold. Again, the extra bold options defined with `BoldFeatures` are appended to the generic font features. Then, the bold font is defined either as the ATS default (`\zf@make@font@shapes'` optional argument is to check if there actually is one; if not, the bold NFSS series is left undefined) or with the font specified with the `BoldFont` feature.

```
216      \unless\ifzf@nobf
217      \ifx\zf@bf\@empty
218      \zf@make@font@shapes[\zf@fontname]{/B}
219      {\bfdefault}{\updefault}{\zf@font@feat\zf@bf@feat}%
220      \else
221      \zf@make@font@shapes{\zf@bf}
222      {\bfdefault}{\updefault}{\zf@font@feat\zf@bf@feat}%
223      \fi
224      \fi
```

And italic in the same way:

```
225      \unless\ifzf@noit
226      \ifx\zf@it\@empty
227      \zf@make@font@shapes[\zf@fontname]{/I}
228      {\mddefault}{\itdefault}{\zf@font@feat\zf@it@feat}%
229      \else
230      \zf@make@font@shapes{\zf@it}
231      {\mddefault}{\itdefault}{\zf@font@feat\zf@it@feat}%
232      \fi
233      \fi
```

If requested, the custom fonts take precedence when choosing the bold italic font. When both italic and bold fonts are requested and the bold italic font hasn't been explicitly specified (a rare occurrence, presumably), the new bold font is used to define the new bold italic font.

```
234      \@tempswatrue
235      \ifzf@nobf\@tempswafalse\fi
236      \ifzf@noit\@tempswafalse\fi
237      \if@tempswa
238      \ifx\zf@bfit\@empty
239      \ifx\zf@bf\@empty
240      \ifx\zf@it\@empty
241      \zf@make@font@shapes[\zf@fontname]{/BI}
242      {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}%
243      \else
244      \zf@make@font@shapes[\zf@it]{/B}
245      {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}%
246      \fi
247      \else
248      \zf@make@font@shapes[\zf@bf]{/I}
249      {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}%
```

```

250     \fi
251   \else
252     \zf@make@font@shapes{\zf@bfit}
253     {\bfdefault}{\itdefault}{\zf@font@feat\zf@bfit@feat}%
254   \fi
255   \fi
256 \fi
257 \endgroup}

```

8.6.1 Fonts

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

```

258 \newcommand*\zf@set@font@type{%
259   \zf@tfmfalse \zf@atsuifalse \zf@icufalse \zf@mmfalse
260   \ifcase\XeTeXfonttype\zf@basefont
261     \zf@tfm
262   \or
263     \zf@atsuitrue
264     \ifnum\XeTeXcountvariations\zf@basefont > 0
265       \zf@mmtrue
266     \fi
267   \or
268     \zf@icutrue
269   \fi}

```

`\zf@make@font@shapes` [**#1**]: Font name prefix

#2 : Font name
 #3 : Font series
 #4 : Font shape
 #5 : Font features

This macro eventually uses `\DeclareFontShape` to define the font shape in question.

The optional first argument is used when making the font shapes for bold, italic, and bold italic fonts using \XeTeX 's auto-recognition with #2 as `/B`, `/I`, and `/BI` font name suffixes. If no such font is found, it falls back to the original font name, in which case this macro doesn't proceed and the font shape is not created for the NFSS.

```

270 \newcommand*\zf@make@font@shapes[5][]{%
271   \begingroup
272     \edef\@tempa{#1}%
273     \unless\ifx\@tempa\@empty
274       \font\@tempfonta="\zf@font@wrap{#1}\zf@sufffix" at \f@size pt
275       \edef\@tempa{\fontname\@tempfonta}%

```

```

276 \fi
277 \font\@tempfontb="\zf@font@wrap{#1#2}\zf@suffix" at \f@size pt
278 \edef\@tempb{\fontname\@tempfontb}%
279 \ifx\@tempa\@tempb
280 \zf@PackageInfo{Could not resolve font #1#2 (it might not exist)}%
281 \else
282 \edef\zf@fontname{#1#2}%
283 \let\zf@basefont\@tempfontb
284 \zf@DeclareFontShape{#3}{#4}{#5}%

```

Next, the small caps are defined. `\zf@make@smallcaps` is used to define the appropriate string for activating small caps in the font, if they exist. If we are defining small caps for the upright shape, then the small caps shape default is used. For an *italic* font, however, the shape parameter is overloaded and we must call italic small caps by their own identifier. See Section 8.8 on page 64 for the code that enables this usage.

```

285 \ifx\zf@sc\@empty
286 \unless\ifzf@nosc
287 \zf@make@smallcaps
288 \unless\ifx\zf@smallcaps\@empty
289 \zf@DeclareFontShape[\zf@smallcaps]{#3}
290 {\ifx#4\itdefault\sidefault\else\scdefault\fi}{#5\zf@sc@feat}%
291 \fi
292 \fi
293 \else
294 \edef\zf@fontname{\zf@sc}%
295 \zf@DeclareFontShape{#3}
296 {\ifx#4\itdefault\sidefault\else\scdefault\fi}{#5\zf@sc@feat}%
297 \fi
298 \fi
299 \endgroup}

```

Note that the test for italics to choose the `\sidefault` shape only works while `\zf@fontspec` passes single tokens to this macro...

`\zf@DeclareFontShape` [#1]: Raw appended font feature

#2 : Font series

#3 : Font shape

#4 : Font features

Wrapper for `\DeclareFontShape`.

```

300 \newcommand\zf@DeclareFontShape[4][]{%
301 \ifx\zf@size@feat\@empty
302 \zf@get@feature@requests{#4}%
303 \edef\zf@font@str{<->\zf@scale"\zf@font@wrap\zf@fontname\zf@suffix:%
304 \zf@pre@ff\zf@ff#1"%}
305 \else

```

Default code, above, sets things up for no optical size fonts or features. On the other hand, loop through `SizeFeatures` arguments, which are of the form

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

```

306 \for\zf@this@size:=\zf@size@feat\do{%
307   \let\zf@size\empty
308   \let\zf@size@fnt\zf@fontname
309   \edef\@tempa{\noexpand
310     \setkeys*[zf]{sizing}{\expandafter\@firstofone\zf@this@size}}%
311   \@tempa
312   \ifx\zf@size\empty\zf@PackageError
313     {Size information must be supplied}
314     {For example, SizeFeatures={Size={8-12},...},...}%
315   \fi
316   \zf@get@feature@requests{#4,\XKV@rm}%
317   \edef\zf@font@str{\zf@font@str <\zf@size>%
318     \zf@scale"\zf@size@fnt\zf@suffix:\zf@pre@ff\zf@ff#1"}}%
319 \fi

```

And finally the actual font shape declaration using \zf@font@str defined above. \zf@adjust is defined in various places to deal with things like the hyphenation character and interword spacing.

```

320 \edef\@tempa{\noexpand
321   \DeclareFontShape{\zf@enc}{\zf@family}{#2}{#3}
322   {\zf@font@str}{\zf@adjust}}%
323 \@tempa

```

This extra stuff for the slanted shape substitution is a little bit awkward, but I'd rather have it here than break out yet another macro. Alternatively, one day I might just redefine \slshape. Why not, eh?

```

324 \edef\@tempa{#3}%
325 \edef\@tempb{\itdefault}%
326 \ifx\@tempa\@tempb
327   \edef\@tempa{\noexpand
328     \DeclareFontShape{\zf@enc}{\zf@family}{#1}{\sldefault}
329     {<->sub*\zf@family/#2/\itdefault}{\zf@adjust}}%
330   \@tempa
331 \fi

```

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

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

```

332 \newcommand*{\zf@update@family}[1]{%
333   \ifzf@firsttime
334     \xdef\zf@family@long{\zf@family@long#1}%
335   \fi

```

8.6.2 Features

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

```

336 \newcommand*\zf@get@feature@requests[1]{%
337   \let\zf@ff      \@empty
338   \let\zf@scale   \@empty
339   \let\zf@adjust  \@empty
340   \edef\@tempa{\noexpand\setkeys[zf]{options}\zf@default@options#1}}%
341   \@tempa}

```

`\zf@init` This functionality has been removed from `\zf@get@feature@requests` because it's no longer the first thing that can affect these things.

```

342 \newcommand*\zf@init{%
343   \zf@icufalse
344   \let\zf@pre@ff      \@empty
345   \let\zf@font@feat   \@empty
346   \let\zf@font@str    \@empty
347   \let\zf@font@wrap   \@firstofone
348   \let\zf@suffix      \@empty
349   \let\zf@bf          \@empty
350   \let\zf@it          \@empty
351   \let\zf@bfit        \@empty
352   \let\zf@sc          \@empty
353   \let\zf@up@feat     \@empty
354   \let\zf@bf@feat     \@empty
355   \let\zf@it@feat     \@empty
356   \let\zf@bfit@feat   \@empty
357   \let\zf@sc@feat     \@empty
358   \let\zf@size        \@empty
359   \let\zf@size@feat   \@empty
360   \let\zf@size@fnt    \@empty
361   \c@zf@script 1818326126\relax
362   \def\zf@script@name{Latin}%
363   \c@zf@language 0\relax
364   \def\zf@language@name{Default}%
365 }

```

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

```

366 \newcommand*\zf@make@smallcaps{%
367   \let\zf@smallcaps\@empty
368   \ifzf@atsui
369     \zf@make@aat@feature@string{3}{3}%
370     \unless\ifx\@tempa\@empty
371       \edef\zf@smallcaps{\@tempa;}%

```

```

372 \fi
373 \fi
374 \ifzf@icu
375 \zf@check@ot@feat{+smcp}%
376 \if@tempswa
377 \edef\zf@smallcaps{+smcp,}%
378 \fi
379 \fi}

```

`\zf@update@ff` `\zf@ff` is the string used to define the list of specific font features. Each time another font feature is requested, this macro is used to add that feature to the list. AAT features are separated by semicolons, OpenType features by commas.

```

380 \newcommand*\zf@update@ff[1]{%
381 \unless\ifzf@firsttime
382 \xdef\zf@ff{\zf@ff #1\ifzf@icu,\else;\fi}%
383 \fi}

```

`\zf@make@feature` This macro is called by each feature key selected, and runs according to which type of font is selected.

```

384 \newcommand*\zf@make@feature[3]{%
For AAT fonts:
385 \ifzf@atsui
386 \def\@tempa{#1}%
387 \ifx\@tempa\@empty
388 \zf@PackageWarning{%
389 '\XKV@tfam=\XKV@tkey' feature not supported
390 for AAT font '\fontname\zf@basefont'}%
391 \else
392 \zf@make@aat@feature@string{#1}{#2}%
393 \ifx\@tempa\@empty
394 \zf@PackageWarning{%
395 AAT feature '\XKV@tfam=\XKV@tkey'
396 (#1,#2) not available in font '\fontname\zf@basefont'}%
397 \else
398 \zf@update@family{+#1,#2}%
399 \zf@update@ff\@tempa
400 \fi
401 \fi
402 \fi
For OpenType fonts:
403 \ifzf@icu
404 \edef\@tempa{#3}%
405 \ifx\@tempa\@empty
406 \zf@PackageWarning{%
407 '\XKV@tfam=\XKV@tkey' feature not supported

```

```

408     for ICU font '\fontname\zf@basefont'}%
409   \else
410     \expandafter\zf@check@ot@feat\expandafter{\@tempa}%
411     \if@tempswa
412       \zf@update@family{#3}%
413       \zf@update@ff{#3}%
414     \else
415       \zf@PackageWarning{%
416         OpenType feature '\XKV@tfam=\XKV@tkey' (#3)
417         not available in font \fontname\zf@basefont, script
418         '\zf@script@name', language '\zf@language@name'}%
419     \fi
420   \fi
421 \fi}

```

`\zf@define@font@feature` These macros are used in order to simplify font feature definition later on.

```

\zf@define@feature@option 422 \newcommand*\zf@define@font@feature[1]{%
423   \define@key[zf]{options}{#1}{\setkeys[zf@feat]{#1}{##1}}}%
424 \newcommand*\zf@define@feature@option[5]{%
425   \define@key[zf@feat]{#1}{#2}[]{\zf@make@feature{#3}{#4}{#5}}}%

```

`\keyval@alias@key` This macro maps one xkeyval key to another.

```

426 \newcommand*\keyval@alias@key[4][KV]{%
427   \let@cc{#1@#2@#4}{#1@#2@#3}%
428   \let@cc{#1@#2@#4@default}{#1@#2@#3@default}}

```

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

```

429 \newcommand*\multi@alias@key[2]{%
430   \key@ifundefined[zf]{preparse}{#1}
431   {\key@ifundefined[zf]{options}{#1}
432     {\zf@PackageError{The feature #1 doesn't appear to be defined}
433     {It looks like you're trying to rename a feature that doesn't exist.}}
434     {\keyval@alias@key[zf]{options}{#1}{#2}}}
435   {\keyval@alias@key[zf]{preparse}{#1}{#2}}}

```

`\zf@make@aat@feature@string` This macro takes the numerical codes for a font feature and creates a specified macro containing the string required in the font definition to turn that feature on or off. Used primarily in `\zf@make@aat@feature`, but also used to check if small caps exists in the requested font (see page 44).

```

436 \newcommand*\zf@make@aat@feature@string[2]{%
437   \edef\@tempa{\XeTeXfeaturename\zf@basefont #1}%
438   \unless\ifx\@tempa\@empty

```

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

```

439   \ifnum\XeTeXisexclusivefeature\zf@basefont #1>0
440     \edef\@tempb{\XeTeXselectorname\zf@basefont #1 #2}%

```

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

```
441 \else
442 \unless\ifodd #2
443 \edef\@tempb{\XeTeXselectorname\zf@basefont #1 #2}%
```

If the selector is *odd*, it corresponds to switching the feature off. But X_YTeX doesn't return a selector string for this number, since the feature is defined for the 'switching on' value. So we need to check the selector of the previous number, and then prefix the feature string with ! to denote the switch.

```
444 \else
445 \edef\@tempb{\XeTeXselectorname\zf@basefont #1 \numexpr#2-1\relax}%
446 \unless\ifx\@tempb\@empty
447 \edef\@tempb{!\@tempb}%
448 \fi
449 \fi
450 \fi
```

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

```
451 \unless\ifx\@tempb\@empty
452 \edef\@tempa{\@tempa=\@tempb}%
453 \else
454 \let\@tempa\@empty
455 \fi
456 \fi}
```

\zf@iv@strnum This macro takes a four character string and converts it to the numerical representation required for X_YTeX OpenType script/language/feature purposes. The output is stored in \@tempcnta.

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

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

It's probable that all OpenType features *are* in fact four characters long, but not impossible that they aren't. So I'll leave the less efficient parsing stage in there even though it's not strictly necessary for now.

```
457 \newcommand\zf@iv@strnum[1]{%
458 \zf@iv@strnum@i#1 \@nil}
459 \def\zf@iv@strnum@i#1 \@nil{%
460 \zf@iv@strnum@ii#1\@empty\@empty\@nil}
461 \def\zf@iv@strnum@ii#1#2#3#4#5\@nil{%
```



```

462 \@tempcnta\z@
463 \@tempcntb`#1\relax
464 \multiply\@tempcntb"1000000\advance\@tempcnta\@tempcntb
465 \@tempcntb`#2
466 \multiply\@tempcntb"10000\advance\@tempcnta\@tempcntb
467 \expandafter\@tempcntb\ifx\@empty#332\else`#3\fi
468 \multiply\@tempcntb"100\advance\@tempcnta\@tempcntb
469 \expandafter\@tempcntb\ifx\@empty#432\else`#4\fi
470 \advance\@tempcnta\@tempcntb}
471 \newcommand\zf@v@strnum[1]{%
472 \expandafter\zf@iv@strnum@i\@gobble#1 \@nil}

```

TODO: convert to \numexpr

`\zf@check@ot@script` This macro takes an OpenType script tag and checks if it exists in the current font. The output boolean is `\@tempswattrue`. `\@tempcnta` is used to store the number corresponding to the script tag string.

```

473 \newcommand\zf@check@ot@script[1]{%
474 \zf@iv@strnum{#1}%
475 \@tempcntb\XeTeXOTcountscripts\zf@basefont
476 \c@zf@index\z@ \@tempswafalse
477 \loop\ifnum\c@zf@index<\@tempcntb
478 \ifnum\XeTeXOTscripttag\zf@basefont\c@zf@index=\@tempcnta
479 \@tempswattrue
480 \c@zf@index\@tempcntb
481 \else
482 \advance\c@zf@index\@ne
483 \fi
484 \repeat}

```

`\zf@check@ot@lang` This macro takes an OpenType language tag and checks if it exists in the current font/script. The output boolean is `\@tempswattrue`. `\@tempcnta` is used to store the number corresponding to the language tag string. The script used is whatever's held in `\c@zf@script`. By default, that's the number corresponding to 'latn'.

```

485 \newcommand\zf@check@ot@lang[1]{%
486 \zf@iv@strnum{#1}%
487 \@tempcntb\XeTeXOTcountlanguages\zf@basefont\c@zf@script
488 \c@zf@index\z@ \@tempswafalse
489 \loop\ifnum\c@zf@index<\@tempcntb
490 \ifnum\XeTeXOTLanguagetag\zf@basefont\c@zf@script\c@zf@index=\@tempcnta
491 \@tempswattrue
492 \c@zf@index\@tempcntb
493 \else
494 \advance\c@zf@index\@ne
495 \fi
496 \repeat}

```

`\zf@check@ot@feat` This macro takes an OpenType feature tag and checks if it exists in the current font/script/language. The output boolean is `\@tempswa`. `\@tempcnta` is used to store the number corresponding to the feature tag string. The script used is whatever's held in `\c@zf@script`. By default, that's the number corresponding to 'latn'. The language used is `\c@zf@language`, by default 0, the 'default language'.

```

497 \newcommand*\zf@check@ot@feat[1]{%
498   \@tempcntb\XeTeXOTcountfeatures\zf@basefont\c@zf@script\c@zf@language
499   \zf@v@strnum{#1}%
500   \c@zf@index\z@ \@tempswafalse
501   \loop\ifnum\c@zf@index<\@tempcntb
502     \ifnum\XeTeXOTfeaturetag\zf@basefont\c@zf@script\c@zf@language\c@zf@index=\@tempcnta
503       \@tempwattrue
504       \c@zf@index\@tempcntb
505     \else
506       \advance\c@zf@index\@ne
507     \fi
508   \repeat}

```

8.7 keyval definitions

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

8.7.1 Pre-parsed features

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

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

```

509 \define@key[zf]{preparse}{ExternalLocation}[]{%
510   \zf@icuttrue
511   \zf@nobftrue\zf@noittrue
512   \gdef\zf@font@wrap##1{[#1##1]}}

```

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

```

513 \define@choicekey[zf]{preparse}{Renderer}{AAT,ICU}{%
514   \edef\zf@suffix{\zf@suffix/#1}%
515   \font\zf@basefont="\zf@font@wrap\zf@fontname\zf@suffix" at \f@size pt
516   \edef\zf@family@long{\zf@family@long +rend:#1}}

```

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

```
517 \define@key[zf]{preparse}{Script}{%
518   \zf@icutrue
519   \edef\zf@suffix{\zf@suffix/ICU}%
520   \font\zf@basefont="\zf@font@wrap\zf@fontname\zf@suffix" at \f@size pt
521   \edef\zf@family@long{\zf@family@long +script:#1}%
522   {\setkeys[zf@feat]{Script}{#1}}}
```

Exactly the same:

```
523 \define@key[zf]{preparse}{Language}{%
524   \zf@icutrue
525   \edef\zf@suffix{\zf@suffix/ICU}%
526   \font\zf@basefont="\zf@font@wrap\zf@fontname\zf@suffix" at \f@size pt
527   \edef\zf@family@long{\zf@family@long +language:#1}%
528   {\setkeys[zf@feat]{Lang}{#1}}}
```

8.7.2 Bold/italic choosing options

The Bold, Italic, and BoldItalic features are for defining explicitly the bold and italic fonts used in a font family. v1.6 introduced arbitrary font features for these shapes (BoldFeatures, etc.), so the names of the shape-selecting options were appended with Font for consistency.

Fonts Bold:

```
529 \define@key[zf]{preparse}{BoldFont}{%
530   \edef\@tempa{#1}%
531   \ifx\@tempa\@empty
532     \zf@nobftrue
533     \edef\zf@family@long{\zf@family@long nobf}%
534   \else
535     \zf@nobffalse
536     \zf@partial@fontname#1\@nil=\zf@bf
537     \edef\zf@family@long{\zf@family@long bf:#1}%
538   \fi}
```

Same for italic:

```
539 \define@key[zf]{preparse}{ItalicFont}{%
540   \edef\@tempa{#1}%
541   \ifx\@tempa\@empty
542     \zf@noittrue
543     \edef\zf@family@long{\zf@family@long noit}%
544   \else
545     \zf@noitfalse
546     \zf@partial@fontname#1\@nil=\zf@it
547     \edef\zf@family@long{\zf@family@long it:#1}%
548   \fi}
```

Simpler for bold+italic:

```
549 \define@key[zf]{preparse}{BoldItalicFont}{%
550   \zf@partial@fontname#1\@nil=\zf@bfit
551   \edef\zf@family@long{\zf@family@long bfit:#1}}
```

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

```
552 \define@key[zf]{options}{SmallCapsFont}{%
553   \edef\@tempa{#1}%
554   \ifx\@tempa\@empty
555     \zf@nosctrue
556     \edef\zf@family@long{\zf@family@long nosc}%
557   \else
558     \zf@noscfalse
559     \zf@partial@fontname#1\@nil=\zf@sc
560     \zf@update@family{sc:\zap@space #1 \@empty}%
561   \fi}
```

`\zf@partial@fontname` This macro takes the first token of its input and ends up defining #3 to the name of the font depending if it's been specified in full ("Baskerville Semibold") or in abbreviation ("* Semibold").

This could be done more flexibly by making * active; I'll change it later if I need to.

```
562 \def\zf@partial@fontname#1#2\@nil=#3{%
563   \if#1*\relax
564     \edef#3{\zf@fontname#2}%
565   \else
566     \edef#3{#1#2}%
567   \fi}
```

Features

```
568 \define@key[zf]{preparse}{UprightFeatures}{%
569   \def\zf@up@feat{, #1}%
570   \edef\zf@family@long{\zf@family@long rmfeat:#1}}
571 \define@key[zf]{preparse}{BoldFeatures}{%
572   \def\zf@bf@feat{, #1}%
573   \edef\zf@family@long{\zf@family@long bfeat:#1}}
574 \define@key[zf]{preparse}{ItalicFeatures}{%
575   \def\zf@it@feat{, #1}%
576   \edef\zf@family@long{\zf@family@long itfeat:#1}}
577 \define@key[zf]{preparse}{BoldItalicFeatures}{%
578   \def\zf@bfit@feat{, #1}%
579   \edef\zf@family@long{\zf@family@long bfitfeat:#1}}
```

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

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

paragraphFeatures varying by size

```

583 \define@key[zf]{preparse}{SizeFeatures}{%
584   \unless\ifzf@firsttime\def\zf@size@feat{#1}\fi
585   \zf@update@family{sizefeat:\zap@space #1 \@empty}}
586 \define@key[zf]{sizing}{Size}{\def\zf@size{#1}}
587 \define@key[zf]{sizing}{Font}{\def\zf@size@fnt{#1}}

```

8.7.3 Font-independent features

These features can be applied to any font.

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

```

588 \define@key[zf]{options}{Scale}{%
589   \edef\@tempa{#1}%
590   \edef\@tempb{MatchLowercase}%
591   \ifx\@tempa\@tempb
592     \zf@calc@scale{5}%
593   \else
594     \edef\@tempb{MatchUppercase}%
595     \ifx\@tempa\@tempb
596       \zf@calc@scale{8}%
597     \else
598       \edef\zf@scale{#1}%
599     \fi
600   \fi
601   \zf@update@family{+scale:\zf@scale}%
602   \edef\zf@scale{s*[\zf@scale]}}

```

`\zf@calc@scale` This macro calculates the amount of scaling between the default roman font and the (default shape of) the font being selected such that the font dimension that is input is equal for both. The only font dimensions that justify this are 5 (lowercase height) and 8 (uppercase height in X_YTEX).

This script is executed for every extra shape, which seems wasteful, but allows alternate italic shapes from a separate font, say, to be loaded and to be auto-scaled correctly. Even if this would be ugly.

```

603 \newcommand\zf@calc@scale[1]{%
604   \begingroup
605     \rmfamily
606     \setlength\@tempdima{\fontdimen#1\font}%
607     \setlength\@tempdimb{\fontdimen#1\zf@basefont}%
608     \setlength\@tempdimc{1pt*\ratio{\@tempdima}{\@tempdimb}}%
609     \xdef\zf@scale{\strip@pt\@tempdimc}%
610     \zf@PackageInfo{\zf@fontname\space scale = \zf@scale}%
611   \endgroup}

```

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

```
612 \define@key[zf]{options}{WordSpace}{%
613   \zf@update@family{+wordspace:#1}%
614   \unless\ifzf@firsttime
615     \zf@wordspace@parse#1,\zf@ii,\zf@iii,\zf@
616     \fi}
```

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

```
617 \def\zf@wordspace@parse#1,#2,#3,#4\zf@{%
618   \def\@tempa{#4}%
619   \ifx\@tempa\empty
620     \setlength\@tempdima{#1\fontdimen2\zf@basefont}%
621     \@tempdimb\@tempdima
622     \@tempdimc\@tempdima
623   \else
624     \setlength\@tempdima{#1\fontdimen2\zf@basefont}%
625     \setlength\@tempdimb{#2\fontdimen3\zf@basefont}%
626     \setlength\@tempdimc{#3\fontdimen4\zf@basefont}%
627   \fi
628   \edef\zf@adjust{\zf@adjust
629     \fontdimen2\font\the\@tempdima
630     \fontdimen3\font\the\@tempdimb
631     \fontdimen4\font\the\@tempdimc}}
```

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

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

Letterspacing

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

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

```
639 \define@key[zf]{options}{HyphenChar}{%
640   \zf@update@family{+hyphenchar:#1}%
641   \edef\@tempa{#1}%
642   \edef\@tempb{None}%
```

```

643 \ifx\@tempa\@tempb
644   \g@addto@macro\zf@adjust{\hyphenchar\font-1\relax}%
645 \else
646   \zf@check@one@char#1\zf@@
647   \ifx\@tempb\@empty
648     {\zf@basefont\expandafter\ifnum\expandafter\XeTeXcharglyph\expandafter`#1 > 0
649       \g@addto@macro\zf@adjust{%
650         {\expandafter\hyphenchar\expandafter
651           \font\expandafter`#1}}%
652     \else
653       \zf@PackageError
654       {\fontname\zf@basefont\space doesn't appear to have the glyph cor-
        responding to #1.}
655       {You can't hyphenate with a character that's not available!}%
656     \fi}%
657 \else
658   {\zf@basefont\ifnum\XeTeXcharglyph#1 > 0
659     \g@addto@macro\zf@adjust{\hyphenchar\font#1\relax}%
660   \else
661     \zf@PackageError
662     {\fontname\zf@basefont\space doesn't appear to have the glyph cor-
        responding to #1.}
663     {You can't hyphenate with a character that's not available!}%
664   \fi}%
665 \fi
666 \fi}
667 \def\zf@check@one@char#1#2\zf@@{\def\@tempb{#2}}

```

Colour

```

668 \define@key[zf]{options}{Colour}{%
669   \zf@update@family{+col:#1}%
670   \zf@update@ff{color=#1}}
671 \keyval@alias@key[zf]{options}{Colour}{Color}

```

Mapping

```

672 \define@key[zf]{options}{Mapping}{%
673   \zf@update@family{+map:#1}%
674   \zf@update@ff{mapping=#1}}

```

8.7.4 Continuous font axes

```

675 \define@key[zf]{options}{Weight}{%
676   \zf@update@family{+weight:#1}%
677   \zf@update@ff{weight=#1}}
678 \define@key[zf]{options}{Width}{%
679   \zf@update@family{+width:#1}%

```

```

680 \zf@update@ff{width=#1}}
681 \define@key[zf]{options}{OpticalSize}{%
682 \ifzf@icu
683 \edef\zf@suffix{\zf@suffix/S=#1}%
684 \zf@update@family{+size:#1}%
685 \fi
686 \ifzf@mm
687 \zf@update@family{+size:#1}%
688 \zf@update@ff{optical size=#1}%
689 \fi
690 \ifzf@icu\else
691 \ifzf@mm\else
692 \ifzf@firsttime
693 \zf@PackageWarning
694 {\fontname\zf@basefont\space doesn't appear to have an Opti-
cal Size axis}%
695 \fi
696 \fi
697 \fi}

```

8.7.5 Ligatures

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

```

698 \zf@define@font@feature{Ligatures}
699 \zf@define@feature@option{Ligatures}{Required}      {1}{0}{+rlig}
700 \zf@define@feature@option{Ligatures}{NoRequired}   {1}{1}{-rlig}
701 \zf@define@feature@option{Ligatures}{Common}       {1}{2}{+liga}
702 \zf@define@feature@option{Ligatures}{NoCommon}    {1}{3}{-liga}
703 \zf@define@feature@option{Ligatures}{Rare}         {1}{4}{+dlig}
704 \zf@define@feature@option{Ligatures}{NoRare}       {1}{5}{-dlig}
705 \zf@define@feature@option{Ligatures}{Discretionary}{1}{4}{+dlig}
706 \zf@define@feature@option{Ligatures}{NoDiscretionary}{1}{5}{-dlig}
707 \zf@define@feature@option{Ligatures}{Contextual}   {}{} {+clig}
708 \zf@define@feature@option{Ligatures}{NoContextual} {}{} {-clig}
709 \zf@define@feature@option{Ligatures}{Historical}   {}{} {+hlig}
710 \zf@define@feature@option{Ligatures}{NoHistorical} {}{} {-hlig}
711 \zf@define@feature@option{Ligatures}{Logos}        {1}{6} {}
712 \zf@define@feature@option{Ligatures}{NoLogos}     {1}{7} {}
713 \zf@define@feature@option{Ligatures}{Rebus}        {1}{8} {}
714 \zf@define@feature@option{Ligatures}{NoRebus}     {1}{9} {}
715 \zf@define@feature@option{Ligatures}{Diphthong}   {1}{10} {}
716 \zf@define@feature@option{Ligatures}{NoDiphthong} {1}{11} {}
717 \zf@define@feature@option{Ligatures}{Squared}     {1}{12} {}
718 \zf@define@feature@option{Ligatures}{NoSquared}   {1}{13} {}
719 \zf@define@feature@option{Ligatures}{AbbrevSquared}{1}{14} {}

```



```

720 \zf@define@feature@option{Ligatures}{NoAbbrevSquared}{1}{15}{ }
721 \zf@define@feature@option{Ligatures}{Icelandic} {1}{32}{ }
722 \zf@define@feature@option{Ligatures}{NoIcelandic} {1}{33}{ }

```

8.7.6 Letters

```

723 \zf@define@font@feature{Letters}
724 \zf@define@feature@option{Letters}{Normal} {3}{0}{ }
725 \zf@define@feature@option{Letters}{Uppercase} {3}{1}{+case}
726 \zf@define@feature@option{Letters}{Lowercase} {3}{2}{ }
727 \zf@define@feature@option{Letters}{SmallCaps} {3}{3}{+smcp}
728 \zf@define@feature@option{Letters}{PetiteCaps} { } { } {+pcap}
729 \zf@define@feature@option{Letters}{UppercaseSmallCaps} { } { } {+c2sc}
730 \zf@define@feature@option{Letters}{UppercasePetiteCaps} { } { } {+c2pc}
731 \zf@define@feature@option{Letters}{InitialCaps} {3}{4}{ }
732 \zf@define@feature@option{Letters}{Unicase} { } { } {+unic}

```

8.7.7 Numbers

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

Both naming conventions are offered to select the number case.

```

733 \zf@define@font@feature{Numbers}
734 \zf@define@feature@option{Numbers}{Monospaced} {6}{0}{+tnum}
735 \zf@define@feature@option{Numbers}{Proportional} {6}{1}{+pnum}
736 \zf@define@feature@option{Numbers}{Lowercase} {21}{0}{+onum}
737 \zf@define@feature@option{Numbers}{OldStyle} {21}{0}{+onum}
738 \zf@define@feature@option{Numbers}{Uppercase} {21}{1}{+lnum}
739 \zf@define@feature@option{Numbers}{Lining} {21}{1}{+lnum}
740 \zf@define@feature@option{Numbers}{SlashedZero} {14}{5}{+zero}
741 \zf@define@feature@option{Numbers}{NoSlashedZero} {14}{4}{-zero}

```

8.7.8 Contextuals

```

742 \zf@define@font@feature {Contextuals}
743 \zf@define@feature@option{Contextuals}{Swash} { } { } {+csw}
744 \zf@define@feature@option{Contextuals}{NoSwash} { } { } {-csw}
745 \zf@define@feature@option{Contextuals}{WordInitial} {8}{0}{+init}
746 \zf@define@feature@option{Contextuals}{NoWordInitial} {8}{1}{-init}
747 \zf@define@feature@option{Contextuals}{WordFinal} {8}{2}{+fina}
748 \zf@define@feature@option{Contextuals}{NoWordFinal} {8}{3}{-fina}
749 \zf@define@feature@option{Contextuals}{LineInitial} {8}{4}{ }
750 \zf@define@feature@option{Contextuals}{NoLineInitial} {8}{5}{ }
751 \zf@define@feature@option{Contextuals}{LineFinal} {8}{6}{+falt}
752 \zf@define@feature@option{Contextuals}{NoLineFinal} {8}{7}{-falt}
753 \zf@define@feature@option{Contextuals}{Inner} {8}{8}{+medi}
754 \zf@define@feature@option{Contextuals}{NoInner} {8}{9}{-medi}

```

8.7.9 Diacritics

```
755 \zf@define@font@feature{Diacritics}
756 \zf@define@feature@option{Diacritics}{Show}      {9}{0}{}
757 \zf@define@feature@option{Diacritics}{Hide}      {9}{1}{}
758 \zf@define@feature@option{Diacritics}{Decompose}{9}{2}{}

```

8.7.10 Kerning

```
759 \zf@define@font@feature{Kerning}
760 \zf@define@feature@option{Kerning}{Uppercase}{}{}{+csp}
761 \zf@define@feature@option{Kerning}{On}          {}{}{+kern}
762 \zf@define@feature@option{Kerning}{Off}         {}{}{-kern}
763 %\zf@define@feature@option{Kerning}{Vertical}{}{}{+vkern}
764 %\zf@define@feature@option{Kerning}{VerticalAlternateProportional}{}{}{+vpal}
765 %\zf@define@feature@option{Kerning}{VerticalAlternateHalfWidth} {}{}{+vhal}

```

8.7.11 Vertical position

```
766 \zf@define@font@feature{VerticalPosition}
767 \zf@define@feature@option{VerticalPosition}{Normal} {10}{0}{}
768 \zf@define@feature@option{VerticalPosition}{Superior} {10}{1}{+sup}
769 \zf@define@feature@option{VerticalPosition}{Inferior} {10}{2}{+sub}
770 \zf@define@feature@option{VerticalPosition}{Ordinal} {10}{3}{+ordn}
771 \zf@define@feature@option{VerticalPosition}{Numerator} {} {} {+numr}
772 \zf@define@feature@option{VerticalPosition}{Denominator} {} {} {+dnom}
773 \zf@define@feature@option{VerticalPosition}{ScientificInferior}{}{}{+sinf}

```

8.7.12 Fractions

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

```

8.7.13 Alternates and variants

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

```
779 \define@key[zf]{options}{Alternate}{%
780   \setkeys*[zf@feat]{Alternate}{#1}%
781   \unless\ifx\XKV@rm\@empty
782     \def\XKV@tfam{Alternate}%
783     \zf@make@feature{17}{#1}{}%
784   \fi}

785 \define@key[zf]{options}{Variant}{%
786   \setkeys*[zf@feat]{Variant}{#1}%
787   \unless\ifx\XKV@rm\@empty
788     \def\XKV@tfam{Variant}%

```

```

789 \zf@make@feature{18}{#1}{+ss\two@digits{#1}}%
790 \fi}

```

8.7.14 Style

```

791 \zf@define@font@feature{Style}
792 \zf@define@feature@option{Style}{Alternate}      {} {} {+salt}
793 \zf@define@feature@option{Style}{Italic}          {32}{2}{+ital}
794 \zf@define@feature@option{Style}{Ruby}            {28}{2}{+ruby}
795 \zf@define@feature@option{Style}{Swash}           {} {} {+swsh}
796 \zf@define@feature@option{Style}{Historic}        {} {} {+hist}
797 \zf@define@feature@option{Style}{Display}         {19}{1}{}
798 \zf@define@feature@option{Style}{Engraved}        {19}{2}{}
799 \zf@define@feature@option{Style}{TitlingCaps}     {19}{4}{+titl}
800 \zf@define@feature@option{Style}{TallCaps}        {19}{5}{}
801 \zf@define@feature@option{Style}{HorizontalKana} {} {} {+hkna}
802 \zf@define@feature@option{Style}{VerticalKana}   {} {} {+vkna}

```

8.7.15 CJK shape

```

803 \zf@define@font@feature{CJKShape}
804 \zf@define@feature@option{CJKShape}{Traditional}{20}{0} {+trad}
805 \zf@define@feature@option{CJKShape}{Simplified}  {20}{1} {+smpl}
806 \zf@define@feature@option{CJKShape}{JIS1978}    {20}{2} {+jp78}
807 \zf@define@feature@option{CJKShape}{JIS1983}    {20}{3} {+jp83}
808 \zf@define@feature@option{CJKShape}{JIS1990}    {20}{4} {+jp90}
809 \zf@define@feature@option{CJKShape}{Expert}     {20}{10}{+expt}
810 \zf@define@feature@option{CJKShape}{NLC}        {20}{13}{+nlck}

```

8.7.16 Character width

```

811 \zf@define@font@feature{CharacterWidth}
812 \zf@define@feature@option{CharacterWidth}{Proportional}{22}{0}{+pwid}
813 \zf@define@feature@option{CharacterWidth}{Full}{22}{1}{+fwid}
814 \zf@define@feature@option{CharacterWidth}{Half}{22}{2}{+hwid}
815 \zf@define@feature@option{CharacterWidth}{Third}{22}{3}{+twid}
816 \zf@define@feature@option{CharacterWidth}{Quarter}{22}{4}{+qwid}
817 \zf@define@feature@option{CharacterWidth}{AlternateProportional}{22}{5}{+palt}
818 \zf@define@feature@option{CharacterWidth}{AlternateHalf}{22}{6}{+halt}
819 \zf@define@feature@option{CharacterWidth}{Default}{22}{7}{}

```

8.7.17 Annotation

```

820 \zf@define@font@feature{Annotation}
821 \zf@define@feature@option{Annotation}{Off}{24}{0}{-nalt}
822 \zf@define@feature@option{Annotation}{On}{}{} {+nalt}
823 \zf@define@feature@option{Annotation}{Box}{24}{1}{}
824 \zf@define@feature@option{Annotation}{RoundedBox}{24}{2}{}
825 \zf@define@feature@option{Annotation}{Circle}{24}{3}{}
826 \zf@define@feature@option{Annotation}{BlackCircle}{24}{4}{}

```

```

827 \zf@define@feature@option{Annotation}{Parenthesis}{24}{5}{}
828 \zf@define@feature@option{Annotation}{Period}{24}{6}{}
829 \zf@define@feature@option{Annotation}{RomanNumerals}{24}{7}{}
830 \zf@define@feature@option{Annotation}{Diamond}{24}{8}{}
831 \zf@define@feature@option{Annotation}{BlackSquare}{24}{9}{}
832 \zf@define@feature@option{Annotation}{BlackRoundSquare}{24}{10}{}
833 \zf@define@feature@option{Annotation}{DoubleCircle}{24}{11}{}

```

8.7.18 Vertical

```

834 \zf@define@font@feature{Vertical}
835 \define@key[zf@feat]{Vertical}{RotatedGlyphs}[]{}%
836 \ifzf@icu
837 \zf@make@feature{}{}{+vrt2}%
838 \else
839 \zf@update@family{+vert}%
840 \zf@update@ff{vertical}%
841 \fi}

```

8.7.19 Script

```

842 \newfontscript{Arabic}{arab} \newfontscript{Armenian}{armn}
843 \newfontscript{Balinese}{bali} \newfontscript{Bengali}{beng}
844 \newfontscript{Bopomofo}{bopo} \newfontscript{Braille}{brai}
845 \newfontscript{Buginese}{bugi} \newfontscript{Buhid}{buhd}
846 \newfontscript{Byzantine Music}{byzm} \newfontscript{Canadian Syllab-
ics}{cans}
847 \newfontscript{Cherokee}{cher}
848 \newfontscript{CJK Ideographic}{hani} \newfontscript{Coptic}{copt}
849 \newfontscript{Cypriot Syllabary}{cpri} \newfontscript{Cyrillic}{cyr1}
850 \newfontscript{Default}{DFLT} \newfontscript{Deseret}{dsrt}
851 \newfontscript{Devanagari}{deva} \newfontscript{Ethiopic}{ethi}
852 \newfontscript{Georgian}{geor} \newfontscript{Glagolitic}{glag}
853 \newfontscript{Gothic}{goth} \newfontscript{Greek}{grek}
854 \newfontscript{Gujarati}{gujr} \newfontscript{Gurmukhi}{guru}
855 \newfontscript{Hangul Jamo}{jamo} \newfontscript{Hangul}{hang}
856 \newfontscript{Hanunoo}{hano} \newfontscript{Hebrew}{hebr}
857 \newfontscript{Hiragana and Katakana}{kana}
858 \newfontscript{Javanese}{java} \newfontscript{Kannada}{knda}
859 \newfontscript{Kharosthi}{khar} \newfontscript{Khmer}{khmr}
860 \newfontscript{Lao}{lao} \newfontscript{Latin}{latn}
861 \newfontscript{Limbu}{limb} \newfontscript{Linear B}{linb}
862 \newfontscript{Malayalam}{mlym} \newfontscript{Math}{math}
863 \newfontscript{Mongolian}{mong}
864 \newfontscript{Musical Symbols}{musc} \newfontscript{Myanmar}{mymr}
865 \newfontscript{N'ko}{nko} \newfontscript{Ogham}{ogam}
866 \newfontscript{Old Italic}{ital} \newfontscript{Old Persian Cuneiform}{xpeo}
867 \newfontscript{Oriya}{orya} \newfontscript{Osmanya}{osma}
868 \newfontscript{Phags-pa}{phag} \newfontscript{Phoenician}{phnx}

```

```

869 \newfontscript{Runic}{runr} \newfontscript{Shavian}{shaw}
870 \newfontscript{Sinhala}{sinh} \newfontscript{Sumero-Akkadian Cuneiform}{xsux}
871 \newfontscript{Syloti Nagri}{sylo} \newfontscript{Syriac}{syr}
872 \newfontscript{Tagalog}{tglg} \newfontscript{Tagbanwa}{tagb}
873 \newfontscript{Tai Le}{tale} \newfontscript{Tai Lu}{tal}
874 \newfontscript{Tamil}{taml} \newfontscript{Telugu}{telu}
875 \newfontscript{Thaana}{thaa} \newfontscript{Thai}{thai}
876 \newfontscript{Tibetan}{tib} \newfontscript{Tifinagh}{tfng}
877 \newfontscript{Ugaritic Cuneiform}{ugar} \newfontscript{Yi}{yi}

```

8.7.20 Language

```

878 \newfontlanguage{Abaza}{ABA} \newfontlanguage{Abkhazian}{ABK} \newfontlanguage{Adyghe}{ADY}
879 \newfontlanguage{Afrikaans}{AFK} \newfontlanguage{Afar}{AFR} \newfontlanguage{Agaw}{AGW}
880 \newfontlanguage{Altai}{ALT} \newfontlanguage{Amharic}{AMH} \newfontlanguage{Arabic}{ARA}
881 \newfontlanguage{Aari}{ARI} \newfontlanguage{Arakanese}{ARK} \newfontlanguage{Assamese}{ASM}
882 \newfontlanguage{Athapaskan}{ATH} \newfontlanguage{Avar}{AVR} \newfontlanguage{Awadhi}{AWA}
883 \newfontlanguage{Aymara}{AYM} \newfontlanguage{Azeri}{AZE} \newfontlanguage{Badaga}{BAD}
884 \newfontlanguage{Baghelkhandi}{BAG} \newfontlanguage{Balkar}{BAL} \newfontlanguage{Baule}{BAU}
885 \newfontlanguage{Berber}{BBR} \newfontlanguage{Bench}{BCH} \newfontlanguage{Bible Cree}{BCR}
886 \newfontlanguage{Belarussian}{BEL} \newfontlanguage{Bemba}{BEM} \newfontlanguage{Bengali}{BEN}
887 \newfontlanguage{Bulgarian}{BGR} \newfontlanguage{Bhili}{BHI} \newfontlanguage{Bhojpuri}{BHO}
888 \newfontlanguage{Bikol}{BIK} \newfontlanguage{Bilen}{BIL} \newfontlanguage{Blackfoot}{BKF}
889 \newfontlanguage{Balochi}{BLI} \newfontlanguage{Balante}{BLN} \newfontlanguage{Balti}{BLT}
890 \newfontlanguage{Bambara}{BMB} \newfontlanguage{Bamileke}{BML} \newfontlanguage{Breton}{BRE}
891 \newfontlanguage{Brahui}{BRH} \newfontlanguage{Braj Bhasha}{BRI} \newfontlanguage{Burmese}{BRM}
892 \newfontlanguage{Bashkir}{BSH} \newfontlanguage{Beti}{BTI} \newfontlanguage{Catalan}{CAT}
893 \newfontlanguage{Cebuano}{CEB} \newfontlanguage{Chechen}{CHE} \newfontlanguage{Chaha Gurage}{CHG}
894 \newfontlanguage{Chattisgarhi}{CHH} \newfontlanguage{Chichewa}{CHI} \newfontlanguage{Chukchi}{CHU}
895 \newfontlanguage{Chipewyan}{CHP} \newfontlanguage{Cherokee}{CHR} \newfontlanguage{Chuvash}{CHV}
896 \newfontlanguage{Comorian}{CMR} \newfontlanguage{Coptic}{COP} \newfontlanguage{Cree}{CRE}
897 \newfontlanguage{Carrier}{CRR} \newfontlanguage{Crimean Tatar}{CRT} \newfontlanguage{Church Slav}
898 \newfontlanguage{Czech}{CSY} \newfontlanguage{Danish}{DAN} \newfontlanguage{Dargwa}{DAR}
899 \newfontlanguage{Woods Cree}{DCR} \newfontlanguage{German}{DEU} \newfontlanguage{Default}{DFT}
900 \newfontlanguage{Dogri}{DGR} \newfontlanguage{Divehi}{DIV} \newfontlanguage{Djerma}{DJR}
901 \newfontlanguage{Dangme}{DNG} \newfontlanguage{Dinka}{DNK} \newfontlanguage{Dungan}{DUN}
902 \newfontlanguage{Dzongkha}{DZN} \newfontlanguage{Ebira}{EBI} \newfontlanguage{Eastern Cree}{ECR}
903 \newfontlanguage{Edo}{EDO} \newfontlanguage{Efik}{EFI} \newfontlanguage{Greek}{ELL}
904 \newfontlanguage{English}{ENG} \newfontlanguage{Erzya}{ERZ} \newfontlanguage{Spanish}{ESP}
905 \newfontlanguage{Estonian}{ETI} \newfontlanguage{Basque}{EUQ} \newfontlanguage{Evenki}{EVK}
906 \newfontlanguage{Even}{EVN} \newfontlanguage{Ewe}{EWE} \newfontlanguage{French Antillean}{FAN}
907 \newfontlanguage{Farsi}{FAR} \newfontlanguage{Finnish}{FIN} \newfontlanguage{Fijian}{FJI}
908 \newfontlanguage{Flemish}{FLE} \newfontlanguage{Forest Nenets}{FNE} \newfontlanguage{Fon}{FON}
909 \newfontlanguage{Faroese}{FOS} \newfontlanguage{French}{FRA} \newfontlanguage{Frisian}{FRI}
910 \newfontlanguage{Friulian}{FRL} \newfontlanguage{Futa}{FTA} \newfontlanguage{Fulani}{FUL}
911 \newfontlanguage{Ga}{GAD} \newfontlanguage{Gaelic}{GAE} \newfontlanguage{Gagauz}{GAG}
912 \newfontlanguage{Galician}{GAL} \newfontlanguage{Garshuni}{GAR} \newfontlanguage{Garhwali}{GAW}

```

913 \newfontlanguage{Ge'ez}{GEZ}\newfontlanguage{Gilyak}{GIL}\newfontlanguage{Gumuz}{GMZ}
 914 \newfontlanguage{Gondi}{GON}\newfontlanguage{Greenlandic}{GRN}\newfontlanguage{Garo}{GRO}
 915 \newfontlanguage{Guarani}{GUA}\newfontlanguage{Gujarati}{GUJ}\newfontlanguage{Haitian}{HAI}
 916 \newfontlanguage{Halami}{HAL}\newfontlanguage{Harauti}{HAR}\newfontlanguage{Hausa}{HAU}
 917 \newfontlanguage{Hawaii}{HAW}\newfontlanguage{Hammer-Banna}{HBN}\newfontlanguage{Hiligaynon}
 918 \newfontlanguage{Hindi}{HIN}\newfontlanguage{High Mari}{HMA}\newfontlanguage{Hindko}{HND}
 919 \newfontlanguage{Ho}{HO}\newfontlanguage{Harari}{HRI}\newfontlanguage{Croatian}{HRV}
 920 \newfontlanguage{Hungarian}{HUN}\newfontlanguage{Armenian}{HYE}\newfontlanguage{Igbo}{IBO}
 921 \newfontlanguage{Ijo}{IJO}\newfontlanguage{Ilokano}{ILO}\newfontlanguage{Indonesian}{IND}
 922 \newfontlanguage{Ingush}{ING}\newfontlanguage{Inuktitut}{INU}\newfontlanguage{Irish}{IRI}
 923 \newfontlanguage{Irish Traditional}{IRT}\newfontlanguage{Icelandic}{ISL}\newfontlanguage{Inar
 924 \newfontlanguage{Italian}{ITA}\newfontlanguage{Hebrew}{IWR}\newfontlanguage{Javanese}{JAV}
 925 \newfontlanguage{Yiddish}{JII}\newfontlanguage{Japanese}{JAN}\newfontlanguage{Judezmo}{JUD}
 926 \newfontlanguage{Jula}{JUL}\newfontlanguage{Kabardian}{KAB}\newfontlanguage{Kachchi}{KAC}
 927 \newfontlanguage{Kalenjin}{KAL}\newfontlanguage{Kannada}{KAN}\newfontlanguage{Karachay}{KAR}
 928 \newfontlanguage{Georgian}{KAT}\newfontlanguage{Kazakh}{KAZ}\newfontlanguage{Kebena}{KEB}
 929 \newfontlanguage{Khutsuri Georgian}{KGE}\newfontlanguage{Khakass}{KHA}\newfontlanguage{Khanty-
 Kazim}{KHK}
 930 \newfontlanguage{Khmer}{KHM}\newfontlanguage{Khanty-Shurishkar}{KHS}\newfontlanguage{Khanty-
 Vakhi}{KHV}
 931 \newfontlanguage{Khowar}{KHW}\newfontlanguage{Kikuyu}{KIK}\newfontlanguage{Kirghiz}{KIR}
 932 \newfontlanguage{Kisii}{KIS}\newfontlanguage{Kokni}{KKN}\newfontlanguage{Kalmyk}{KLM}
 933 \newfontlanguage{Kamba}{KMB}\newfontlanguage{Kumaoni}{KMN}\newfontlanguage{Komo}{KMO}
 934 \newfontlanguage{Komso}{KMS}\newfontlanguage{Kanuri}{KNR}\newfontlanguage{Kodagu}{KOD}
 935 \newfontlanguage{Korean Old Hangul}{KOH}\newfontlanguage{Konkani}{KOK}\newfontlanguage{Kikongo}
 936 \newfontlanguage{Komi-Permyak}{KOP}\newfontlanguage{Korean}{KOR}\newfontlanguage{Komi-
 Zyrian}{KOZ}
 937 \newfontlanguage{Kpelle}{KPL}\newfontlanguage{Krio}{KRI}\newfontlanguage{Karakalpak}{KRK}
 938 \newfontlanguage{Karelian}{KRL}\newfontlanguage{Karaim}{KRM}\newfontlanguage{Karen}{KRN}
 939 \newfontlanguage{Koorote}{KRT}\newfontlanguage{Kashmiri}{KSH}\newfontlanguage{Khasi}{KSI}
 940 \newfontlanguage{Kildin Sami}{KSM}\newfontlanguage{Kui}{KUI}\newfontlanguage{Kulvi}{KUL}
 941 \newfontlanguage{Kumyk}{KUM}\newfontlanguage{Kurdish}{KUR}\newfontlanguage{Kurukh}{KUU}
 942 \newfontlanguage{Kuy}{KUY}\newfontlanguage{Koryak}{KYK}\newfontlanguage{Ladin}{LAD}
 943 \newfontlanguage{Lahuli}{LAH}\newfontlanguage{Lak}{LAK}\newfontlanguage{Lambani}{LAM}
 944 \newfontlanguage{Lao}{LAO}\newfontlanguage{Latin}{LAT}\newfontlanguage{Laz}{LAZ}
 945 \newfontlanguage{L-Cree}{LCR}\newfontlanguage{Ladakhi}{LDK}\newfontlanguage{Lezgi}{LEZ}
 946 \newfontlanguage{Lingala}{LIN}\newfontlanguage{Low Mari}{LMA}\newfontlanguage{Limbu}{LMB}
 947 \newfontlanguage{Lomwe}{LMW}\newfontlanguage{Lower Sorbian}{LSB}\newfontlanguage{Lule Sami}{LS
 948 \newfontlanguage{Lithuanian}{LTH}\newfontlanguage{Luba}{LUB}\newfontlanguage{Luganda}{LUG}
 949 \newfontlanguage{Luhya}{LUH}\newfontlanguage{Luo}{LUO}\newfontlanguage{Latvian}{LVI}
 950 \newfontlanguage{Majang}{MAJ}\newfontlanguage{Makua}{MAK}\newfontlanguage{Malayalam Tra-
 ditional}{MAL}
 951 \newfontlanguage{Mansi}{MAN}\newfontlanguage{Marathi}{MAR}\newfontlanguage{Marwari}{MAW}
 952 \newfontlanguage{Mbundu}{MBN}\newfontlanguage{Manchu}{MCH}\newfontlanguage{Moose Cree}{MCR}
 953 \newfontlanguage{Mende}{MDE}\newfontlanguage{Me'en}{MEN}\newfontlanguage{Mizo}{MIZ}
 954 \newfontlanguage{Macedonian}{MKD}\newfontlanguage{Male}{MLE}\newfontlanguage{Malagasy}{MLG}
 955 \newfontlanguage{Malinke}{MLN}\newfontlanguage{Malayalam Reformed}{MLR}\newfontlanguage{Malay

956 \newfontlanguage{Mandinka}{MND}\newfontlanguage{Mongolian}{MNG}\newfontlanguage{Manipuri}{MNI}
 957 \newfontlanguage{Maninka}{MNK}\newfontlanguage{Manx Gaelic}{MNX}\newfontlanguage{Moksha}{MOK}
 958 \newfontlanguage{Moldavian}{MOL}\newfontlanguage{Mon}{MON}\newfontlanguage{Moroccan}{MOR}
 959 \newfontlanguage{Maori}{MRI} \newfontlanguage{Maithili}{MTH} \newfontlanguage{Maltese}{MTS}
 960 \newfontlanguage{Mundari}{MUN} \newfontlanguage{Naga-Assamese}{NAG} \new-
 fontlanguage{Nanai}{NAN}
 961 \newfontlanguage{Naskapi}{NAS} \newfontlanguage{N-Cree}{NCR} \newfontlanguage{Ndebele}{NDB}
 962 \newfontlanguage{Ndonga}{NDG} \newfontlanguage{Nepali}{NEP} \newfontlanguage{Newari}{NEW}
 963 \newfontlanguage{Nagari}{NGR} \newfontlanguage{Norway House Cree}{NHC} \new-
 fontlanguage{Nisi}{NIS}
 964 \newfontlanguage{Niuean}{NIU} \newfontlanguage{Nkole}{NKL} \newfontlanguage{N'ko}{NKO}
 965 \newfontlanguage{Dutch}{NLD} \newfontlanguage{Nogai}{NOG} \newfontlanguage{Norwegian}{NOR}
 966 \newfontlanguage{Northern Sami}{NSM} \newfontlanguage{Northern Tai}{NTA} \new-
 fontlanguage{Esperanto}{NTO}
 967 \newfontlanguage{Nynorsk}{NYN} \newfontlanguage{Oji-Cree}{OCR} \newfont-
 language{Ojibway}{OJB}
 968 \newfontlanguage{Oriya}{ORI} \newfontlanguage{Oromo}{ORO} \newfontlanguage{Ossetian}{OSS}
 969 \newfontlanguage{Palestinian Aramaic}{PAA} \newfontlanguage{Pali}{PAL} \new-
 fontlanguage{Punjabi}{PAN}
 970 \newfontlanguage{Palpa}{PAP} \newfontlanguage{Pashto}{PAS} \newfontlanguage{Polytonic Greek}{P}
 971 \newfontlanguage{Pilipino}{PIL} \newfontlanguage{Palaung}{PLG} \newfont-
 language{Polish}{PLK}
 972 \newfontlanguage{Provençal}{PRO} \newfontlanguage{Portuguese}{PTG} \new-
 fontlanguage{Chin}{QIN}
 973 \newfontlanguage{Rajasthani}{RAJ} \newfontlanguage{R-Cree}{RCR} \newfont-
 language{Russian Buriat}{RBU}
 974 \newfontlanguage{Riang}{RIA} \newfontlanguage{Rhaeto-Romanic}{RMS} \new-
 fontlanguage{Romanian}{ROM}
 975 \newfontlanguage{Romany}{ROY} \newfontlanguage{Rusyn}{RSY} \newfontlanguage{Ruanda}{RUA}
 976 \newfontlanguage{Russian}{RUS} \newfontlanguage{Sadri}{SAD} \newfontlanguage{Sanskrit}{SAN}
 977 \newfontlanguage{Santali}{SAT} \newfontlanguage{Sayisi}{SAY} \newfontlanguage{Sekota}{SEK}
 978 \newfontlanguage{Selkup}{SEL} \newfontlanguage{Sango}{SGO} \newfontlanguage{Shan}{SHN}
 979 \newfontlanguage{Sibe}{SIB} \newfontlanguage{Sidamo}{SID} \newfontlanguage{Silte Gurage}{SIG}
 980 \newfontlanguage{Skolt Sami}{SKS} \newfontlanguage{Slovak}{SKY} \newfont-
 language{Slavey}{SLA}
 981 \newfontlanguage{Slovenian}{SLV} \newfontlanguage{Somali}{SML} \newfont-
 language{Samoan}{SMO}
 982 \newfontlanguage{Sena}{SNA} \newfontlanguage{Sindhi}{SND} \newfontlanguage{Sinhalese}{SNH}
 983 \newfontlanguage{Soninke}{SNK} \newfontlanguage{Sodo Gurage}{SOG} \new-
 fontlanguage{Sotho}{SOT}
 984 \newfontlanguage{Albanian}{SQI} \newfontlanguage{Serbian}{SRB} \newfont-
 language{Saraiki}{SRK}
 985 \newfontlanguage{Serer}{SRR} \newfontlanguage{South Slavey}{SSL} \newfont-
 language{Southern Sami}{SSM}
 986 \newfontlanguage{Suri}{SUR} \newfontlanguage{Svan}{SVA} \newfontlanguage{Swedish}{SVE}
 987 \newfontlanguage{Swadaya Aramaic}{SWA} \newfontlanguage{Swahili}{SWK} \new-
 fontlanguage{Swazi}{SWZ}

```

988 \newfontlanguage{Sutu}{SXT} \newfontlanguage{Syriac}{SYR} \newfontlanguage{Tabasaran}{TAB}
989 \newfontlanguage{Tajiki}{TAJ} \newfontlanguage{Tamil}{TAM} \newfontlanguage{Tatar}{TAT}
990 \newfontlanguage{TH-Cree}{TCR} \newfontlanguage{Telugu}{TEL} \newfontlanguage{Tongan}{TGN}
991 \newfontlanguage{Tigre}{TGR} \newfontlanguage{Tigrinya}{TGY} \newfontlanguage{Thai}{THA}
992 \newfontlanguage{Tahitian}{THT} \newfontlanguage{Tibetan}{TIB} \newfont-
    language{Turkmen}{TKM}
993 \newfontlanguage{Temne}{TMN} \newfontlanguage{Tswana}{TNA} \newfontlanguage{Tundra Nenets}{TNE}
994 \newfontlanguage{Tonga}{TNG} \newfontlanguage{Todo}{TOD}
995 \newfontlanguage{Tsonga}{TSG} \newfontlanguage{Turoyo Aramaic}{TUA} \new-
    fontlanguage{Tulu}{TUL}
996 \newfontlanguage{Tuvini}{TUV} \newfontlanguage{Twili}{TWI} \newfontlanguage{Udmurt}{UDM}
997 \newfontlanguage{Ukrainian}{UKR} \newfontlanguage{Urdu}{URD} \newfontlanguage{Upper Sor-
    bian}{USB}
998 \newfontlanguage{Uyghur}{UYG} \newfontlanguage{Uzbek}{UZB} \newfontlanguage{Venda}{VEN}
999 \newfontlanguage{Vietnamese}{VIT} \newfontlanguage{Wa}{WA} \newfontlanguage{Wagdi}{WAG}
1000 \newfontlanguage{West-Cree}{WCR} \newfontlanguage{Welsh}{WEL} \newfontlanguage{Wolof}{WLF}
1001 \newfontlanguage{Tai Lue}{XBD} \newfontlanguage{Xhosa}{XHS} \newfontlanguage{Yakut}{YAK}
1002 \newfontlanguage{Yoruba}{YBA} \newfontlanguage{Y-Cree}{YCR} \newfontlanguage{Yi Clas-
    sic}{YIC}
1003 \newfontlanguage{Yi Modern}{YIM} \newfontlanguage{Chinese Hong Kong}{ZHH}
1004 \newfontlanguage{Chinese Phonetic}{ZHP} \newfontlanguage{Chinese Simpli-
    fied}{ZHS}
1005 \newfontlanguage{Chinese Traditional}{ZHT} \newfontlanguage{Zande}{ZND} \new-
    fontlanguage{Zulu}{ZUL}

```

Turkish Turns out that many fonts use ‘TUR’ as their Turkish language tag rather than the specified ‘TRK’. So we check for both:

```

1006 \define@key[zf@feat]{Lang}{Turkish}[]{}%
1007   \zf@check@ot@lang{TRK}%
1008   \if@tempswa
1009     \c@zf@language\@tempcnta\relax
1010     \xdef\zf@language@name{Turkish}%
1011     \xdef\zf@family@long{\zf@family@long+lang=Turkish}%
1012     \xdef\zf@pre@ff{\zf@pre@ff language=TRK,}%
1013   \else
1014     \zf@check@ot@lang{TUR}%
1015     \if@tempswa
1016       \c@zf@language\@tempcnta\relax
1017       \xdef\zf@language@name{Turkish}%
1018       \xdef\zf@family@long{\zf@family@long+lang=Turkish}%
1019       \xdef\zf@pre@ff{\zf@pre@ff language=TUR,}%
1020     \else
1021       \zf@PackageWarning{Font \fontname\zf@basefont does not contain
1022         language '#1' for script '\zf@script@name'}%
1023     \fi
1024   \fi}

```


8.7.21 Raw feature string

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

```
1025 \define@key[zf]{options}{RawFeature}{%  
1026   \zf@update@family{+Raw:#1}%  
1027   \zf@update@ff{#1}}
```

8.8 Italic small caps

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

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

```
1028 \providecommand*\sidefault{si}  
1029 \DeclareRobustCommand\sishape{%  
1030   \not@math@alphabet\sishape\relax  
1031   \fontshape\sidefault\selectfont}  
1032 \DeclareTextFontCommand\textsi{\sishape}
```

`\zf@merge@shape` This is the macro which enables the overload on the `\. .shape` commands. It takes three such arguments. In essence, the macro selects the first argument, unless the second argument is already selected, in which case it selects the third.

```
1033 \newcommand*\zf@merge@shape}[3]{%  
1034   \edef\@tempa{#1}%  
1035   \edef\@tempb{#2}%  
1036   \ifx\@tempa\@tempb  
1037     \ifcsname\@tempa\encoding/\@family/\@series/#3\endcsname  
1038     \edef\@tempa{#3}%  
1039     \fi  
1040   \fi  
1041   \fontshape{\@tempa}\selectfont}
```

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

```
\upshape 1042 \DeclareRobustCommand\itshape{%  
1043   \not@math@alphabet\itshape\mathit  
1044   \zf@merge@shape\itdefault\scdefault\sidefault}  
1045 \DeclareRobustCommand\slshape{%  
1046   \not@math@alphabet\slshape\relax  
1047   \zf@merge@shape\sldefault\scdefault\sidefault}
```

```

1048 \DeclareRobustCommand{\scshape}{%
1049   \not@math@alphabet\scshape\relax
1050   \zf@merge@shape\scdefault\itdefault\sidefault}
1051 \DeclareRobustCommand{\upshape}{%
1052   \not@math@alphabet\upshape\relax
1053   \zf@merge@shape\updefault\sidefault\scdefault}

```

\em Redefinitions moved to the xltextra package.
\emph

8.9 Selecting maths fonts

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

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

Since every maths setup is slightly different, we also take different paths for defining various math glyphs depending which maths font package has been loaded. As far as I am aware, the only two options for \LaTeX are euler and lucb-math. Unless I've got all confused and the mathtime fonts are not virtual fonts either. But I'm pretty sure they are.

```

1054 \@ifpackageloaded{euler}{\zf@package@euler@loadedtrue}
1055                           {\zf@package@euler@loadedfalse}
1056 \def\zf@math{%
1057   \let\zf@font@warning\@font@warning
1058   \let\@font@warning\@font@info
1059   \@ifpackageloaded{euler}{%
1060     \ifzf@package@euler@loaded
1061       \zf@math@eulertrue
1062     \else
1063       \zf@PackageError{The euler package must be loaded BEFORE fontspec}
1064       {fontspec only overwrites euler's attempt to \MessageBreak
1065       define the maths text fonts if fontspec is \MessageBreak
1066       loaded after euler. Type <return> to proceed\MessageBreak
1067       with incorrect \protect\mathit, \protect\mathbf, etc}
1068     \fi}{}
1069   \@ifpackageloaded{lucbmath}{\zf@math@lucidatrue}{}
1070   \@ifpackageloaded{lucidabr}{\zf@math@lucidatrue}{}
1071   \@ifpackageloaded{lucimatx}{\zf@math@lucidatrue}{}

```

Knuth's CM fonts are all squashed together, combining letters, accents, text symbols and maths symbols all in the one font, `cmr`, plus other things in other fonts. Because we are changing the roman font in the document, we need to redefine all of the maths glyphs in \LaTeX 's operators maths font to still go back to the

legacy cmr font for all these random glyphs, unless a separate maths font package has been loaded instead.

In every case, the maths accents are always taken from the operators font, which is generally the main text font. (Actually, there is a `\hat` accent in Euler-Fraktur, but it's *ugly*. So I ignore it. Sorry if this causes inconvenience.)

```

1072 \DeclareSymbolFont{legacymaths}{OT1}{cmr}{m}{n}
1073 \SetSymbolFont{legacymaths}{bold}{OT1}{cmr}{bx}{n}
1074 \DeclareMathAccent{\acute}{\mathalpha}{legacymaths}{19}
1075 \DeclareMathAccent{\grave}{\mathalpha}{legacymaths}{18}
1076 \DeclareMathAccent{\ddot}{\mathalpha}{legacymaths}{127}
1077 \DeclareMathAccent{\tilde}{\mathalpha}{legacymaths}{126}
1078 \DeclareMathAccent{\bar}{\mathalpha}{legacymaths}{22}
1079 \DeclareMathAccent{\breve}{\mathalpha}{legacymaths}{21}
1080 \DeclareMathAccent{\check}{\mathalpha}{legacymaths}{20}
1081 \DeclareMathAccent{\hat}{\mathalpha}{legacymaths}{94} % too bad, euler
1082 \DeclareMathAccent{\dot}{\mathalpha}{legacymaths}{95}
1083 \DeclareMathAccent{\mathring}{\mathalpha}{legacymaths}{23}

```

`\colon`: **what's going on?** Okay, so `:` and `\colon` in maths mode are defined in a few places, so I need to work out what does what. Respectively, we have:

```

% fontmath.ltx:
\DeclareMathSymbol{\colon}{\mathpunct}{operators}{"3A}
\DeclareMathSymbol{:}{\mathrel}{operators}{"3A}

% amsmath.sty:
\renewcommand{\colon}{\nobreak\mskip2mu\mathpunct{}\nonscript
\mkern-\thinmuskip{:}\mskip6mu\plus1mu\relax}

% euler.sty:
\DeclareMathSymbol{:}{\mathrel}{EulerFraktur}{"3A}

% lucbmath.sty:
\DeclareMathSymbol{\@tempb}{\mathpunct}{operators}{58}
\ifx\colon\@tempb
\DeclareMathSymbol{\colon}{\mathpunct}{operators}{58}
\fi
\DeclareMathSymbol{:}{\mathrel}{operators}{58}

```

($3A_{16} = 58_{10}$) So I think, based on this summary, that it is fair to tell fontspec to 'replace' the operators font with legacymaths for this symbol, except when `amsmath` is loaded since we want to keep its definition.

```

1084 \begingroup
1085 \mathchardef\@tempa="603A %
1086 \let\next\egroup

```

```

1087 \ifx\colon\@tempa
1088 \DeclareMathSymbol{\colon}{\mathpunct}{legacymaths}{58}
1089 \fi
1090 \endgroup

```

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

```

1091 \ifzf@math@euler\else
1092 \DeclareMathSymbol{!}{\mathclose}{legacymaths}{33}
1093 \DeclareMathSymbol{:}{\mathrel}{legacymaths}{58}
1094 \DeclareMathSymbol{;}{\mathpunct}{legacymaths}{59}
1095 \DeclareMathSymbol{?}{\mathclose}{legacymaths}{63}

```

And these ones are defined both in euler and lucbmath, so we only need to run this code if no extra maths package has been loaded.

```

1096 \ifzf@math@lucida\else
1097 \DeclareMathSymbol{0}{\mathalpha}{legacymaths}{0}
1098 \DeclareMathSymbol{1}{\mathalpha}{legacymaths}{1}
1099 \DeclareMathSymbol{2}{\mathalpha}{legacymaths}{2}
1100 \DeclareMathSymbol{3}{\mathalpha}{legacymaths}{3}
1101 \DeclareMathSymbol{4}{\mathalpha}{legacymaths}{4}
1102 \DeclareMathSymbol{5}{\mathalpha}{legacymaths}{5}
1103 \DeclareMathSymbol{6}{\mathalpha}{legacymaths}{6}
1104 \DeclareMathSymbol{7}{\mathalpha}{legacymaths}{7}
1105 \DeclareMathSymbol{8}{\mathalpha}{legacymaths}{8}
1106 \DeclareMathSymbol{9}{\mathalpha}{legacymaths}{9}
1107 \DeclareMathSymbol{\Gamma}{\mathalpha}{legacymaths}{10}
1108 \DeclareMathSymbol{\Delta}{\mathalpha}{legacymaths}{11}
1109 \DeclareMathSymbol{\Theta}{\mathalpha}{legacymaths}{12}
1110 \DeclareMathSymbol{\Lambda}{\mathalpha}{legacymaths}{13}
1111 \DeclareMathSymbol{\Xi}{\mathalpha}{legacymaths}{14}
1112 \DeclareMathSymbol{\Pi}{\mathalpha}{legacymaths}{15}
1113 \DeclareMathSymbol{\Sigma}{\mathalpha}{legacymaths}{16}
1114 \DeclareMathSymbol{\Upsilon}{\mathalpha}{legacymaths}{17}
1115 \DeclareMathSymbol{\Phi}{\mathalpha}{legacymaths}{18}
1116 \DeclareMathSymbol{\Psi}{\mathalpha}{legacymaths}{19}
1117 \DeclareMathSymbol{\Omega}{\mathalpha}{legacymaths}{20}
1118 \DeclareMathSymbol{+}{\mathbin}{legacymaths}{43}
1119 \DeclareMathSymbol{=}{\mathrel}{legacymaths}{61}
1120 \DeclareMathDelimiter{(}{\mathopen}{legacymaths}{40}{largesymbols}{0}
1121 \DeclareMathDelimiter{)}{\mathclose}{legacymaths}{41}{largesymbols}{1}
1122 \DeclareMathDelimiter{[}{\mathopen}{legacymaths}{91}{largesymbols}{2}
1123 \DeclareMathDelimiter{]}{\mathclose}{legacymaths}{93}{largesymbols}{3}
1124 \DeclareMathDelimiter{/}{\mathord}{legacymaths}{47}{largesymbols}{14}
1125 \DeclareMathSymbol{\mathdollar}{\mathord}{legacymaths}{36}
1126 \fi
1127 \fi

```

Finally, we change the font definitions for `\mathrm` and so on. These are defined using the `\zf@rmmaths (...)` macros, which default to `\rmdefault` but may be specified with the `\setmathrm (...)` commands in the preamble.

Since L^AT_EX only generally defines one level of boldness, we omit `\mathbf` in the bold maths series. It can be specified as per usual with `\setboldmathrm`, which stores the appropriate family name in `\zf@rmboldmaths`.

```

1128 \DeclareSymbolFont{operators}\zf@enc\zf@rmmaths\mddefault\updefault
1129 \SetSymbolFont{operators}{normal}\zf@enc\zf@rmmaths\mddefault\updefault
1130 \SetMathAlphabet\mathrm{normal}\zf@enc\zf@rmmaths\mddefault\updefault
1131 \SetMathAlphabet\mathit{normal}\zf@enc\zf@rmmaths\mddefault\itdefault
1132 \SetMathAlphabet\mathbf{normal}\zf@enc\zf@rmmaths\bfdefault\updefault
1133 \SetMathAlphabet\mathsf{normal}\zf@enc\zf@sfmaths\mddefault\updefault
1134 \SetMathAlphabet\mathtt{normal}\zf@enc\zf@ttmaths\mddefault\updefault
1135 \SetSymbolFont{operators}{bold}\zf@enc\zf@rmmaths\bfdefault\updefault
1136 \ifdefined\zf@rmboldmaths
1137 \SetMathAlphabet\mathrm{bold}\zf@enc\zf@rmboldmaths\mddefault\updefault
1138 \SetMathAlphabet\mathbf{bold}\zf@enc\zf@rmboldmaths\bfdefault\updefault
1139 \SetMathAlphabet\mathit{bold}\zf@enc\zf@rmboldmaths\mddefault\itdefault
1140 \else
1141 \SetMathAlphabet\mathrm{bold}\zf@enc\zf@rmmaths\bfdefault\updefault
1142 \SetMathAlphabet\mathit{bold}\zf@enc\zf@rmmaths\bfdefault\itdefault
1143 \fi
1144 \SetMathAlphabet\mathsf{bold}\zf@enc\zf@sfmaths\bfdefault\updefault
1145 \SetMathAlphabet\mathtt{bold}\zf@enc\zf@ttmaths\bfdefault\updefault
1146 \let\font@warning\zf@font@warning

```

`\zf@math@maybe` We're a little less sophisticated about not executing the `\zf@maths` macro if various other maths font packages are loaded. This list is based on the wonderful 'L^AT_EX Font Catalogue': <http://www.tug.dk/FontCatalogue/mathfonts.html>. I'm sure there are more I've missed. Do the T_EX Gyre fonts have maths support yet?

Untested: would `\unless\ifnum\Gamma=28672\relax\zf@mathfalse\fi` be a better test? This needs more cooperation with euler and lucida, I think.

```

1147 \def\zf@math@maybe{%
1148 \ifpackageloaded{anttor}{\ifx\define@antt@mathversions a\zf@mathfalse\fi}}
1149 \ifpackageloaded{arev}{\zf@mathfalse}}
1150 \ifpackageloaded{eulervm}{\zf@mathfalse}}
1151 \ifpackageloaded{mathdesign}{\zf@mathfalse}}
1152 \ifpackageloaded{concmath}{\zf@mathfalse}}
1153 \ifpackageloaded{cmbright}{\zf@mathfalse}}
1154 \ifpackageloaded{mathesf}{\zf@mathfalse}}
1155 \ifpackageloaded{gfsartemis}{\zf@mathfalse}}
1156 \ifpackageloaded{gfsneoellenic}{\zf@mathfalse}}
1157 \ifpackageloaded{iwona}{\ifx\define@iwona@mathversions a\zf@mathfalse\fi}}
1158 \ifpackageloaded{kpfonts}{\zf@mathfalse}}
1159 \ifpackageloaded{kmath}{\zf@mathfalse}}

```

```

1160 \ifpackageloaded{kurier}{\ifx\define@kurier@mathversions a\zf@mathfalse\fi}{}
1161 \ifpackageloaded{fouriernc}{\zf@mathfalse}{}
1162 \ifpackageloaded{fourier}{\zf@mathfalse}{}
1163 \ifpackageloaded{mathpazo}{\zf@mathfalse}{}
1164 \ifpackageloaded{mathptmx}{\zf@mathfalse}{}
1165 \ifpackageloaded{unicode-math}{\zf@mathfalse}{}
1166 \ifzf@math
1167     \zf@PackageWarning{Adjusting the maths setup^^} (use [no-
        math] to avoid this)}
1168 \zf@math
1169 \fi}
1170 \AtBeginDocument{\zf@math@maybe}

```

8.10 Finishing up

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

```

1171 \ifzf@configfile
1172 \InputIfFileExists{fontspec.cfg}
1173 {\typeout{fontspec.cfg loaded.}}
1174 {\typeout{No fontspec.cfg file found; no configuration loaded.}}
1175 \fi

```

The end! Thanks for coming.

File II

fontspec.cfg

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

```
1
2 %%%%%%%%%%
3 %% FOR BACKWARDS COMPATIBILITY WITH PREVIOUS VERSIONS %%
4
5 \let\newfontinstance\newfontfamily
6
7 \newcommand\newfeaturecode[3]{%
8   \define@key{zf}{#1}[]{\zf@make@feature{#2}{#3}{}}
9
10 \aliasfontfeature{BoldFont}{Bold}
11 \aliasfontfeature{ItalicFont}{Italic}
12 \aliasfontfeature{BoldItalicFont}{BoldItalic}
13 \aliasfontfeature{SmallCapsFont}{SmallCaps}
14 \aliasfontfeature{Style}{StyleOptions}
15 \aliasfontfeature{Contextuals}{Swashes}
16 \aliasfontfeatureoption{Contextuals}{Swash}{Contextual}
17 \aliasfontfeatureoption{Letters}{UppercaseSmallCaps}{SMALLCAPS}
18 \aliasfontfeatureoption{Letters}{UppercasePetiteCaps}{PETITECAPS}
19
20 %%%%%%%%%%
21 %% FOR CONVENIENCE %%
22
23 \newfontscript{Kana}{kana}
24 \newfontscript{Maths}{math}
25 \newfontscript{CJK}{hani}
26
```

File III

fontspec-example.ltx

```
1 \documentclass{article}
2
3 \usepackage{euler}
4 \usepackage[cm-default]{fontspec}
5 \usepackage{xltextra}
6
7 \defaultfontfeatures{Scale=MatchLowercase,Mapping=tex-text}
8 \setmainfont[Numbers=Lowercase]{FPL Neu}
```

```

9\setsansfont{Lucida Sans}
10\setmonofont{Lucida Sans Typewriter}
11
12\frenchspacing % TeX's default is a little old-fashioned...
13
14\begin{document}
15\pagestyle{empty}
16
17\section*{The basics of the \textsf{fontspec} package}
18
19The \textsf{fontspec} package enables automatic font selection for \La-
TeX{} documents typeset with \XeTeX{}. The basic command is\
20\indent \verb|\fontspec[font features]{font display name}|.\
21As an example:
22
23\begin{center}
24  \Large
25  \fontspec[
26    Colour          = 0000CC,
27    Numbers          = OldStyle,
28    VerticalPosition = Ordinal,
29    Variant          = 2
30    ]{Apple Chancery}
31  My 1st example of Apple Chancery
32\end{center}
33
34The default, sans serif, and typewriter fonts may be set with the \verb|\setmainfont|, \verb|\set
mands, respectively, as shown in the preamble. They take the same syn-
tax as the \verb|\fontspec| package. All expected font shapes are available:
35
36\begin{center}
37  {\itshape Italics and \scshape small caps\dots}\
38  {\sffamily\bfseries Bold sans serif and \itshape bold italic sans serif\dots}
39\end{center}
40
41With the roman and sans serif fonts set in the preamble, text fonts in math mode are also changed:
face 'Euler' has been used in this document (with the \textsf{euler} pack-
age---or the \textsf{eulervm} package if the \lpxpdfvixml driver is be-
ing used), since the default Computer Modern maths font is rather light.
42\[
43  \mathcal{F}(s) = \int^{\infty}_0 f(t) \exp(-st)\,\mathrm{d}t
44\]
45
46You'll also notice the \verb|\defaultfontfeatures| command in the pream-
ble. This command takes a single argument of font features that are then ap-
plied to every subsequent instance of font selection. The first argu-
ment in this case, \verb|\Mapping=tex-text|, enables regular \TeX{} liga-

```


tures like `\verb|``---'|` for ```---'`. The second automatically scales the fonts to the same x-height.

⁴⁷

⁴⁸Please see the documentation for font feature explanation and further package niceties.

⁴⁹

⁵⁰`\end{document}`

Change History

v1.0	
General: Initial version.	31
v1.1	
General: Name change to fontspec.	31
\setmainfont: Implemented (with friends).	34
v1.10	
General: Color brought back into the .sty	54
New feature LetterSpace.	53
Some babel encoding problems resolved.	33
\addfontfeatures: Saved family information macro changes.	36
\AtBeginDocument: Added lucimatx checking. (Not really tested, though.)	65
Fixed Lucida bug (missing \else)	65
\zf@fontspec: Saved family info split into two (now three) macros.	41
Space zapped from L ^A T _E X family name due to various problems.	41
\zf@make@feature: Removed embarrassing space after warnings.	46
v1.11	
General: HyphenChar checks its input now.	53
Added better support for Turkish language selection.	64
Ensure bold/italic fonts are loaded with the same renderer as the regular font even if unspecified.	41
OpenType Variant fixed.	57
\emph: Redefinitions moved to xltextra.	65
\newfontface: Name change from \newfontfamily.	35
\newfontlanguage: Fixed \c@zf@language setting not being global.	37
\newfontscript: Fixed \c@zf@script setting not being global.	37
\zf@wordspace@parse: Improved saving \fontdimen stuff to \zd@adjust(also see PunctuationSpace).	53
v1.12	
General: BoldFont, etc., flags \zf@nobf conditional false rather than assuming it implicitly. This allows, <i>e.g.</i> , empty BoldFont to be overloaded.	50
Finally, use the EU1 font encoding.	33
New feature ExternalLocation for loading external fonts.	49
Package option for disabling the EU1 encoding.	32
\addfontfeatures: Now use grouping to restore \zf@default@option change.	36
\zf@make@aat@feature@string: Fixed result of \XeTeXfeaturename output change (empty string if odd non-exclusive selector).	47
Removed \@thisfontfeature macro; replaced with \@tempa.	47
v1.13	
General: RawFeature added	64
SizeFeatures: Beginnings of support for different font features for different font sizes, required by unimath.	52
\zf@DeclareFontShape: New feature SizeFeatures implemented.	43
\zf@make@feature: Warns (and fixes bug) when an AAT feature is requested for an ICU font and vice versa.	46

\zf@partial@fontname: Folded in the \@tempa\let command, changing the syntax.	51
v1.2	
General: Initial OpenType support.	31
Support for Scale.	52
v1.3	
General: More OpenType support.	31
Support for Mapping and Colour.	54
\defaultfontfeatures: Implemented.	35
\newAATfeature: Implemented.	36
\newfontfeature: Implemented.	36
v1.3a	
General: Bug fix for OpenType small caps.	56
v1.4	
General: Support for Weight and Width AAT features.	54
\AtBeginDocument: Selects the default \mathXX fonts.	65
\defaultfontfeatures: Name changed from \setdefaultoptions.	35
v1.5	
General: New options for arbitrary bold/italic shapes.	50
\addfontfeatures: Implemented.	36
\zf@fontspec: Added code for choosing arbitrary bold/italic fonts.	41
Checks if the font family has already been defined.	41
NFSS specifiers now take the default values.	41
\zf@make@font@shapes: Absorbed font-checking from \zf@fontspec.	42
v1.5a	
\AtBeginDocument: Added fix for Computer Modern maths.	65
v1.6	
General: Bold option aliased to BoldFont.	50
LetterCase is now Letters and options changed appropriately.	56
Scale feature now updates family name.	52
All AAT Fractions features offered.	57
New OpenType feature: Language	60
New OpenType feature: Script	59
OpenType letters features: PetiteCaps and PETITECAPS.	56
OpenType ligature features: Contextual and Historical.	55
OpenType stylistic sets supports under the Variant option.	57
\addfontfeatures: Removed \relaxing of temporary macros.	36
\AtBeginDocument: Removed mathtime support since XeTeX doesn't handle virtual fonts. Why did I put it in in the first place?	65
\fontspec: Removed \zf@currfont (unnecessary)	33
\newfontface: Implemented.	35
\newfontfeature: newff counter now uses LaTeX methods rather than primitive TeX. I don't know if there is any advantage to this.	36
\setmainfont: Changed \rmdefault, etc., assigning to use \let directly.	34
\zf@fontspec: Added code for choosing arbitrary bold/italic font features.	41
Writes some info to the .log file	41

\zf@get@feature@requests: Removed the space between the comma and \zf@options when it's concatenated with the defaults.	44
v1.7	
General: Style feature renamed from StyleOptions.	58
AAT Numbers:SlashedZero.	56
New feature: Annotation	58
New feature: CharacterShape	58
New feature: CharacterWidth	58
New feature: OpticalSize; works with both OpenType and MM fonts.	54
OpenType Alternate Fractions feature.	57
OpenType Alternate now only AAT.	57
Removed AAT check for weight/width axes (could also be Multiple Master)	54
\zf@define@feature@option: Implemented for the bulk of the feature processing code.	46
\zf@fontspec: Optional argument now mandatory.	41
\zf@make@aat@feature@string: Changed some \edefs to \let	47
Removed third argument; always saves the feature string in \zf@thisfontfeature	47
\zf@make@feature: Accommodation of the \zf@thisfontfeature change.	46
\zf@make@font@shapes: Changed some \edefs to \let.	42
Support for the OpticalSize feature.	42
\zf@make@smallcaps: Accommodation of the \zf@thisfontfeature change.	45
\zf@set@font@type: Added 'MM' font type; tests true, e.g., with Skia & Minion MM. Used with the OpticalSize feature.	41
Removed exclusivity from font type (AAT, OpenType) check, since fonts can be both.	41
Removed various \count255s.	41
\zf@update@ff: Fix for featureless fonts (e.g., the MS fonts) being ignored.	45
v1.8	
\AtBeginDocument: Added support for user-specified \mathrm and others.	65
Finally fixed legacy maths font issues. Also checks that euler.sty is loaded in the right order.	65
\setmathrm: Implemented (with friends).	34
v1.8a	
\AtBeginDocument: Added conditional to \colon math symbol (incompatibility with lucida and amsmath)	65
v1.9	
General: CharacterShape now CJKShape	58
SMALLCAPS option changed to UppercaseSmallCaps to facilitate option normalisation (to come). Similarly for PETITECAPS.	56
Swashes feature changed to Contextuals. Option of this feature Contextual changed to Swash, for obvious reasons.	56
TextSpacing now CharacterWidth, with associated option names' change.	58
Alternate/Variant options can be assigned names.	57
New Scale options: MatchLowercase and MatchUppercase.	52
New feature HyphenChar.	53
New feature Kerning.	57
New feature PunctuationSpace.	53

New feature UprightFeatures.	50
New feature Vertical.	59
New feature WordSpace.	53
New features SmallCapsFont and SmallCapsFeatures.	50
Package options (no)config, quiet implemented.	68
\addfontfeatures: Added \ignorespaces to make it invisible.	36
Changed \fontspec call to \@fontspec so that \ignorespaces isn't called unnecessarily.	36
\aliasfontfeature: Implemented.	37
\aliasfontfeatureoption: Implemented.	37
\AtBeginDocument: Maths hex numbers converted to decimal.	65
Suppresses harmless maths font encoding warnings.	68
\emph: Redefined \em in order for nested emphases to work.	65
\fontspec: Added \ignorespaces to make it invisible.	33
\keyval@alias@key: Implemented.	46
\multi@alias@key: Implemented for \aliasfontfeature.	46
\newAATfeature: Replacement for \newfeaturecode.	36
\newfontlanguage: Implemented.	37
\newfontscript: Implemented.	37
\newICUfeature: Implemented.	36
\zf@calc@scale: Implemented for auto-scaling options.	53
\zf@check@ot@feat: Implemented.	49
\zf@check@ot@lang: Implemented.	48
\zf@check@ot@script: Implemented.	48
\zf@DeclareFontShape: Implemented as wrapper for \DeclareFontShape.	43
Slanted/italic shape substitution implemented.	43
\zf@fontspec: Absorbed the comma into \zf@. .@options as to be more efficient when they are not defined.	41
Abstracted the long family name so the NFSS family is simple.	41
Incorporated \zf@get@feature@requests argument change.	41
Incorporated \zf@make@font@shapes change; removed \zf@options storage macro.	41
\zf@get@feature@requests: Absorbed comma into \zf@default@options, making \zf@current@options redundant.	44
Added an argument to eliminate the \zf@options macro.	44
Removed init stuff.	44
\zf@init: Taken from \zf@get@feature@requests.	44
\zf@make@feature: Now checks for OpenType feature.	46
\zf@make@font@shapes: \zf@scale@str eliminated.	42
Absorbed \IfEqFonts.	42
Added argument for \zf@get@feature@requests change.	42
Added code for SmallCaps... features.	42
Added logging of /B, /I, /BI failure.	42
Changed input syntax.	42
Incorporated \sidesdefault test into the \DeclareFontShape argument directly now that it's fully expanded.	42
Made local to hide \zf@fontname changes.	42

Removed \zf@scshape macro.	42
Removed \nfss@catcodes wrapper.	42
\zf@make@smallcaps: Now uses \zf@check@ot@feat.	45
\zf@partial@fontname: Implemented.	51
\zf@update@family: Now fully expands arguments.	43
\zf@update@ff: Removed ridiculous \zf@feature@separator code.	45
\zf@v@strnum: Implemented.	48
\zf@wordspace@parse: Implemented.	53

Index

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

Symbols	
\,	43
\@empty	108, 111, 178, 197, 212, 221, 233–235, 268, 280, 283, 296, 302, 307, 332–334, 339–341, 343–355, 362, 365, 382, 388, 400, 433, 441, 446, 449, 455, 462, 464, 526, 536, 549, 555, 577, 582, 616, 644, 778, 784
\@firstofone	305, 342
\@font@info	1055
\@font@warning	1054, 1055
\@for	301
\@gobble	467
\@ifnextchar	92, 100
\@ifpackageloaded	1051, 1056, 1066–1068
\@nameuse	200
\@ne	477, 489, 501
\@nil	453–456, 467, 531, 541, 545, 554, 557
\@onlypreamble	84–87
\@tempa	95, 98, 103, 106, 112, 116, 172, 267, 268, 270, 274, 304, 306, 315, 318, 319, 321, 322, 325, 335, 336, 365, 366, 381, 382, 388, 394, 399, 400, 405, 432, 433, 447, 449, 525, 526, 535, 536, 548, 549, 586, 588, 592, 615, 616, 638, 640, 1031, 1035, 1038, 1082, 1084
\@tempb	273, 274, 320, 321, 435, 438, 440–442, 446, 447, 587, 588, 591, 592, 639, 640, 644, 664, 1032, 1033
\@tempcnta	147, 158, 457, 459, 461, 463, 465, 473, 485, 497, 1006, 1013
\@tempcntb	458–465, 470, 472, 475, 482, 484, 487, 493, 496, 499
\@tempdima	603, 605, 617–619, 621, 626, 631, 632
\@tempdimb	604, 605, 618, 622, 627
\@tempdimc	605, 606, 619, 623, 628
\@tempfonta	269, 270
\@tempfontb	272, 273, 278
\@tempswafalse	230, 231, 471, 483, 495
\@tempswatru	229, 474, 486, 498
\@zf@configfilefalse	33
\@zf@configfiletrue	32
\@zf@euencfalse	30
\@zf@euenctrue	31
\[.	42
\ \	19, 20, 37
\]	44
A	
\acute	1071
\addfontfeature	9, 120
\addfontfeatures	8, <u>109</u>
\advance	459, 461, 463, 465, 477, 489, 501
\aliasfontfeature	10–15, 29, <u>141</u>
\aliasfontfeatureoption	16–18, 29, <u>141</u>
\AtBeginDocument	<u>1051</u>
B	
\bar	1075
\begin	14, 23, 36
\begingroup	110, 167, 266, 601, 1081
\bfdefault	214, 217, 237, 240, 244, 248, 1129, 1132, 1135, 1138, 1139, 1141, 1142
\bfseries	38
\breve	1076
C	
\c@zf@index	20, 471–473, 475, 477, 483–485, 487, 489, 495–497, 499, 501
\c@zf@language	22, 158, 358, 493, 497, 1006, 1013
\c@zf@newff	19, 123
\c@zf@script	21, 147, 356, 482, 485, 493, 497
\check	1077
\colon	1084, 1085
\cos	41
\csname	26, 27, 29, 114, 115, 125, 198
\cyrillicencoding	50, 53

D	
\ddot	1073
\DeclareFontFamily	208
\DeclareFontShape	316, 323
\DeclareMathAccent	1071–1080
\DeclareMathDelimiter	1117–1121
\DeclareMathSymbol	1085, 1089–1092, 1094–1116, 1122
\DeclareOption	30–34
\DeclareRobustCommand	96, 104, 1026, 1039, 1042, 1045, 1048
\DeclareSymbolFont	1069, 1125
\DeclareTextFontCommand	1029
\def	42, 46, 88–90, 93, 101, 107, 180, 183, 357, 359, 381, 454, 456, 557, 564, 567, 570, 573, 576, 580, 583, 584, 614, 615, 664, 779, 785
\def@cx	26, 123, 196
\defaultfontfeatures	7, 8, 46, 107
\define@choicekey	508
\define@key	8, 124, 144, 155, 418, 420, 504, 512, 518, 524, 534, 544, 547, 563, 566, 569, 572, 575, 578, 583–585, 609, 629, 633, 636, 665, 669, 672, 675, 678, 776, 782, 832, 1003, 1022
\Delta	1105
\do	301
\document	52
\documentclass	1
\dot	1079
\dots	37, 38
E	
\edef	26, 95, 103, 112, 169, 172, 267, 270, 273, 277, 289, 298, 304, 312, 315, 319, 320, 322, 335, 366, 372, 399, 432, 435, 438, 440, 442, 447, 509, 511, 514, 516, 520, 522, 525, 528, 532, 535, 538, 542, 546, 548, 551, 559, 561, 565, 568, 571, 574, 586, 587, 591, 595, 599, 625, 632, 638, 639, 680, 1031, 1032, 1035
\egroup	1083
\else	45, 151, 162, 181, 193, 215, 224, 238, 242, 246, 276, 285, 288, 291, 300, 377, 386, 392, 404, 409, 436, 439, 448, 462, 464, 476, 488, 500, 529, 539, 552, 560, 590, 594, 620, 642, 649, 654, 657, 687, 688, 835, 1010, 1017, 1059, 1088, 1093, 1137
\em	1051
\emph	1051
\encodingdefault	47
\end	32, 39, 50
\endcsname	26, 27, 29, 114, 115, 125, 128, 135, 191, 192, 198, 201, 1034
\endgroup	117, 252, 294, 608, 1087
\ExecuteOptions	35
\exp	43
\expandafter	26, 27, 29, 193, 194, 198, 305, 405, 462, 464, 467, 645, 647, 648
F	
\f@encoding	1034
\f@family	114, 115, 1034
\f@series	1034
\f@shape	1033
\f@size	174, 186, 269, 272, 510, 515, 521
\fi	48, 130, 137, 153, 165, 177, 184, 185, 187, 194, 199, 218, 219, 227, 228, 230, 231, 241, 245, 249–251, 261, 264, 271, 285–287, 291–293, 310, 314, 326, 330, 367, 368, 373, 374, 377, 378, 395–397, 414–416, 443–445, 450, 451, 462, 464, 478, 490, 502, 533, 543, 556, 562, 576, 581, 596, 597, 613, 624, 653, 661–663, 682, 686, 692–694, 781, 787, 838, 1020, 1021, 1036, 1037, 1065, 1086, 1123, 1124, 1140, 1148
\font	174, 186, 269, 272, 510, 515, 521, 603, 626–628, 632, 641, 648, 656
\font@warning	1143
\fontdimen	603, 604, 617, 621–623, 626–628, 631, 632
\fontfamily	57, 97, 105, 118
\fontname	152, 163, 270, 273, 385, 391, 403, 412, 651, 659, 691, 1018
\fontshape	1028, 1038
\fontspec	4, 20, 25, 34, 55
\frenchspacing	12
G	
\g@addto@macro	52, 641, 646, 656

<code>\Gamma</code>	1104	<code>\itdefault</code>	223, 226, 237, 240, 244, 248, 285, 291, 320, 324, 1041, 1047, 1128, 1136, 1139
<code>\gdef</code>	507	<code>\itshape</code>	37, 38, <u>1039</u>
<code>\gdef@cx</code>	<u>26</u> , 204–206		
<code>\global</code>	147, 158		
<code>\grave</code>	1072		
		K	
H		<code>\key@ifundefined</code>	131, 138, 425, 426
<code>\hat</code>	1078	<code>\keyval@alias@key</code>	142, <u>421</u> , 429, 430, 668
<code>\hyphenchar</code>	641, 647, 656		
		L	
I		<code>\Lambda</code>	1107
<code>\if</code>	558	<code>\Large</code>	24
<code>\if@tempswa</code>		<code>\LaTeX</code>	19
. 146, 157, 232, 371, 406, 1005, 1012		<code>\latinencoding</code>	51, 54
<code>\if@zf@configfile</code>	15, 17, 1144	<code>\let</code>	5, 29, 34, 47, 49–51, 53, 54, 61, 63, 66, 70, 74, 77, 80, 83, 108, 111, 120, 170, 173, 278, 302, 303, 332–334, 339–355, 362, 449, 1054, 1055, 1083, 1143
<code>\if@zf@euenc</code>	16, 18, 41	<code>\let@cc</code>	<u>26</u> , 422, 423
<code>\ifcase</code>	255	<code>\loop</code>	472, 484, 496
<code>\ifcsname</code> ..	128, 135, 191, 192, 201, 1034		
<code>\ifdefined</code>	1133		
<code>\ifnum</code>	259, 434, 472, 473, 484, 485, 496, 497, 645, 655		
<code>\ifodd</code>	437	M	
<code>\ifx</code> .	178, 212, 221, 233–235, 268, 274, 280, 283, 285, 291, 296, 307, 321, 365, 382, 388, 400, 433, 441, 446, 462, 464, 526, 536, 549, 588, 592, 616, 640, 644, 778, 784, 1033, 1084	<code>\mathalpha</code>	1071–1080, 1094–1114
<code>\ifzf@atsui</code>	8, 179, 363, 380	<code>\mathbf</code>	1064, 1129, 1135
<code>\ifzf@firsttime</code>		<code>\mathbin</code>	1115
..... 3, 328, 376, 576, 579, 611, 689		<code>\mathcal</code>	43
<code>\ifzf@icu</code>	9, 175, 182, 369, 377, 398, 679, 687, 833	<code>\mathchardef</code>	1082
<code>\ifzf@math@euler</code>	11, 1088	<code>\mathclose</code>	1089, 1092, 1118, 1120
<code>\ifzf@math@lucida</code>	12, 1093	<code>\mathdollar</code>	1122
<code>\ifzf@mm</code>	10, 683, 688	<code>\mathit</code>	1040, 1064, 1128, 1136, 1139
<code>\ifzf@nobf</code>	4, 211, 230	<code>\mathopen</code>	1117, 1119
<code>\ifzf@noit</code>	5, 220, 231	<code>\mathord</code>	1121, 1122
<code>\ifzf@nosc</code>	6, 281	<code>\mathpunct</code>	1085, 1091
<code>\ifzf@package@babel@loaded</code>	14	<code>\mathrel</code>	1090, 1116
<code>\ifzf@package@euler@loaded</code> ..	13, 1057	<code>\mathring</code>	1080
<code>\ifzf@tfm</code>	7	<code>\mathrm</code>	43, 1127, 1134, 1138
<code>\ignorespaces</code>	58, 119	<code>\mathsf</code>	1130, 1141
<code>\indent</code>	20	<code>\mathtt</code>	1131, 1142
<code>\infty</code>	43	<code>\mddefault</code>	210, 223, 226, 1125–1128, 1130, 1131, 1134, 1136
<code>\InputIfFileExists</code>	1145	<code>\MessageBreak</code>	1061–1063
<code>\int</code>	43	<code>\multi@alias@key</code>	141, <u>424</u>
		<code>\multiply</code>	459, 461, 463
		N	
		<code>\newAATfeature</code>	27, <u>127</u>

<code>\newcommand</code>	7, 23–25, 55, 59, 64, 68, 72, 75, 78, 81, 91, 99, 107, 109, 121, 127, 134, 141–143, 154, 166, 253, 265, 295, 327, 331, 337, 361, 375, 379, 417, 419, 421, 424, 431, 452, 466, 468, 480, 492, 600, 1030
<code>\newcount</code>	19–22
<code>\newcounter</code>	194
<code>\newfeaturecode</code>	7
<code>\newfontface</code>	5, <u>91</u>
<code>\newfontface@i</code>	100, 101
<code>\newfontfamily</code>	5, 5, <u>91</u>
<code>\newfontfamily@i</code>	92, 93
<code>\newfontfeature</code>	29, <u>121</u>
<code>\newfontinstance</code>	5
<code>\newfontlanguage</code>	27, <u>154</u> , 875–1002
<code>\newfontscript</code>	23–25, 27, <u>143</u> , 839–874
<code>\newICUfeature</code>	29, <u>134</u>
<code>\newif</code>	3–18
<code>\next</code>	1083
<code>\noexpand</code>	96, 97, 104, 105, 113, 172, 304, 315, 322, 335
<code>\normalfont</code>	62, 67, 71
<code>\not@math@alphabet</code>	1027, 1040, 1043, 1046, 1049
<code>\numexpr</code>	440
O	
<code>\Omega</code>	1114
<code>\or</code>	257, 262
P	
<code>\PackageError</code>	23
<code>\PackageInfo</code>	25
<code>\PackageWarning</code>	24
<code>\pagestyle</code>	15
<code>\Phi</code>	1112
<code>\Pi</code>	1109
<code>\pi</code>	41
<code>\pm</code>	41
<code>\ProcessOptions</code>	36
<code>\protect</code>	1064
<code>\providecommand</code>	26–28, 1025
<code>\Psi</code>	1113
R	
<code>\ratio</code>	605
<code>\relax</code>	147, 158, 356, 358, 440, 458, 558, 641, 656, 1006, 1013, 1027, 1043, 1046, 1049
<code>\repeat</code>	479, 491, 503
<code>\RequirePackage</code>	1, 37, 39, 40, 43, 44
<code>\RequireXeTeX</code>	2, 38
<code>\rmdefault</code>	61, 88
<code>\rmfamily</code>	602
S	
<code>\scdefault</code> 285, 291, 1041, 1044, 1047, 1050	
<code>\scshape</code>	37, <u>1039</u>
<code>\section</code>	17
<code>\selectfont</code>	57, 97, 105, 118, 1028, 1038
<code>\setboldmathrm</code>	6, <u>72</u>
<code>\setkeys</code>	171, 172, 305, 335, 418, 517, 523, 777, 783
<code>\setlength</code>	603–605, 617, 621–623, 631
<code>\setmainfont</code>	5, <u>59</u>
<code>\SetMathAlphabet</code>	1127–1131, 1134–1136, 1138, 1139, 1141, 1142
<code>\setmathrm</code>	6, <u>72</u>
<code>\setmathsf</code>	6, <u>72</u>
<code>\setmathtt</code>	6, <u>72</u>
<code>\setmonofont</code>	5, 10, 34, <u>59</u>
<code>\setromanfont</code>	8, 34, 63
<code>\setsansfont</code>	5, 9, 34, <u>59</u>
<code>\SetSymbolFont</code>	1070, 1126, 1132
<code>\sfdefault</code>	66, 89
<code>\sffamily</code>	38
<code>\sidefont</code>	285, 291, 1025, 1028, 1041, 1044, 1047, 1050
<code>\Sigma</code>	1110
<code>\sishape</code>	<u>1025</u>
<code>\sldefault</code>	323, 1044
<code>\slshape</code>	1042, 1043
<code>\space</code>	607, 651, 659, 691
<code>\stepcounter</code>	122, 193
<code>\strip@pt</code>	606
T	
<code>\TeX</code>	46
<code>\textsf</code>	17, 19, 41
<code>\textsi</code>	<u>1025</u>
<code>\the</code>	123, 198, 626–628, 632
<code>\Theta</code>	1106
<code>\tilde</code>	1074
<code>\ttdefault</code>	70, 90

\two@digits	786	\zf@atsuitrue	258
\typeout	1146, 1147	\zf@basefont	152, 163, 174, 186, 255, 259, 278, 385, 391, 403, 412, 432, 434, 435, 438, 440, 470, 473, 482, 485, 493, 497, 510, 515, 521, 604, 617, 621–623, 631, 645, 651, 655, 659, 691, 1018
U		\zf@bf	212, 216, 234, 243, 344, 531
\unless ..	128, 135, 175, 191, 201, 211, 220, 268, 281, 283, 365, 376, 433, 437, 441, 446, 576, 579, 611, 778, 784	\zf@bf@feat	214, 217, 349, 567
\updefault	210, 214, 217, 1050, 1125–1127, 1129–1132, 1134, 1135, 1138, 1141, 1142	\zf@bf@fit	233, 247, 346, 545
\upshape	1039	\zf@bf@feat	237, 240, 244, 248, 351, 573
\Upsilon	1111	\zf@calc@scale	589, 593, 600
\usepackage	3–5	\zf@check@one@char	643, 664
\UTFencname	49	\zf@check@ot@feat	370, 405, 492
V		\zf@check@ot@lang ..	156, 480, 1004, 1011
\verb	20, 34, 46	\zf@check@ot@script	145, 468
X		\zf@DeclareFontShape ..	279, 284, 290, 295
\xdef	27, 148–150, 159–161, 200, 329, 377, 606, 1007–1009, 1014–1016	\zf@default@options	107, 108, 111, 203, 205, 335
\XeTeX	19	\zf@define@feature@option	133, 140, 417, 696–719, 721–729, 731–738, 740–751, 753–755, 757–762, 764–770, 772–775, 789–799, 801–807, 809–816, 818–830
\XeTeXcharglyph	645, 655	\zf@define@font@feature	129, 136, 417, 695, 720, 730, 739, 752, 756, 763, 771, 788, 800, 808, 817, 831
\XeTeXcountvariations	259	\zf@enc	42, 44, 46, 47, 49–51, 53, 54, 208, 316, 323, 1125–1132, 1134–1136, 1138, 1139, 1141, 1142
\XeTeXfeaturename	432	\zf@family	57, 61, 66, 70, 74, 77, 80, 83, 97, 105, 118, 200, 201, 204–206, 208, 316, 323, 324
\XeTeXfonttype	255	\zf@family@long	149, 160, 170, 191, 196, 200, 329, 511, 516, 522, 528, 532, 538, 542, 546, 551, 565, 568, 571, 574, 1008, 1015
\XeTeXisexclusivefeature	434	\zf@ff	207, 299, 313, 332, 377
\XeTeXOTcountfeatures	493	\zf@firsttimefalse	190
\XeTeXOTcountlanguages	482	\zf@firsttimetrue	188
\XeTeXOTcountscripts	470	\zf@font@feat ...	173, 189, 210, 214, 217, 223, 226, 237, 240, 244, 248, 340
\XeTeXOTfeaturetag	497	\zf@font@str	298, 312, 317, 341
\XeTeXOTlanguagetag	485	\zf@font@warning	1054, 1143
\XeTeXOTscripttag	473	\zf@font@wrap	174, 186, 269, 272, 298, 342, 507, 510, 515, 521
\XeTeXselectorname	435, 438, 440		
\Xi	1108		
\XKV@rm	172, 173, 311, 778, 784		
\XKV@tfam ...	384, 390, 402, 411, 779, 785		
\XKV@tkey	384, 390, 402, 411		
Z			
\z@	457, 471, 483, 495		
\zap@space	197, 555, 577, 582		
\zf@@	612, 614, 643, 664		
\zf@@ii	612		
\zf@@iii	612		
\zf@adjust	317, 324, 334, 625, 632, 641, 646, 656		
\zf@atsuifalse	254		

<code>\zf@fontname</code>	169, 170, 174, 186, 204, 207, 209, 213, 222, 236, 277, 289, 298, 303, 510, 515, 521, 559, 607
<code>\zf@fontspec</code>	56, 60, 65, 69, 73, 76, 79, 82, 94, 102, 113, <u>166</u>
<code>\zf@get@feature@requests</code> 189, 297, 311, <u>331</u>
<code>\zf@icufalse</code>	254, <u>338</u>
<code>\zf@icutrue</code>	263, 505, 513, 519
<code>\zf@init</code>	168, <u>337</u>
<code>\zf@it</code>	221, 225, 235, 239, 345, 541
<code>\zf@it@feat</code>	223, 226, 350, 570
<code>\zf@iv@strnum</code>	452, 469, 481
<code>\zf@iv@strnum@i</code>	453, 454, 467
<code>\zf@iv@strnum@ii</code>	455, 456
<code>\zf@language@name</code> 159, 359, 413, 1007, 1014
<code>\zf@make@aat@feature@string</code> 364, 387, <u>431</u>
<code>\zf@make@feature</code> 8, <u>379</u> , 420, 780, 786, 834	
<code>\zf@make@font@shapes</code>	209, 213, 216, 222, 225, 236, 239, 243, 247, <u>265</u>
<code>\zf@make@smallcaps</code>	282, <u>361</u>
<code>\zf@math@eulertrue</code>	1058
<code>\zf@math@lucidatrue</code>	1066–1068
<code>\zf@merge@shape</code> <u>1030</u> , 1041, 1044, 1047, 1050
<code>\zf@mmfalse</code>	254
<code>\zf@mmtrue</code>	260
<code>\zf@nobffalse</code>	530
<code>\zf@nobftrue</code>	506, 527
<code>\zf@noitfalse</code>	540
<code>\zf@noittrue</code>	506, 537
<code>\zf@noscfalse</code>	553
<code>\zf@nosctrue</code>	550
<code>\zf@package@euler@loadedfalse</code> . .	1052
<code>\zf@package@euler@loadedtrue</code> . .	1051
<code>\zf@PackageError</code> 23, 307, 427, 650, 658, 1060
<code>\zf@PackageInfo</code> . . .	25, 34, 202, 275, 607
<code>\zf@PackageWarning</code> .	24, 34, 132, 139, 152, 163, 383, 389, 401, 410, 690, 1018
<code>\zf@partial@fontname</code> 531, 541, 545, 554, <u>557</u>
<code>\zf@pre@ff</code>	150, 161, 207, 299, 313, 339, 1009, 1016
<code>\zf@rmboldmaths</code>	77, 1133–1136
<code>\zf@rmmaths</code> 74, 88, 1125–1129, 1132, 1138, 1139
<code>\zf@sc</code>	280, 289, 347, 554
<code>\zf@sc@feat</code>	285, 291, 352, 576
<code>\zf@scale</code> 298, 313, 333, 595, 598, 599, 606, 607
<code>\zf@script@name</code>	148, 164, 357, 413, 1019
<code>\zf@set@font@type</code>	176, <u>253</u>
<code>\zf@sfmaths</code>	80, 89, 1130, 1141
<code>\zf@size</code>	302, 307, 312, 353, 583
<code>\zf@size@feat</code>	296, 301, 354, 580
<code>\zf@size@fnt</code>	303, 313, 355, 584
<code>\zf@smallcaps</code> . . .	283, 284, 362, 366, 372
<code>\zf@suffix</code>	174, 178, 180, 183, 186, 207, 269, 272, 298, 313, 343, 509, 510, 514, 515, 520, 521, 680
<code>\zf@tfm</code>	256
<code>\zf@tfmfalse</code>	254
<code>\zf@this@size</code>	301, 305
<code>\zf@ttmaths</code>	83, 90, 1131, 1142
<code>\zf@up@feat</code>	210, 348, 564
<code>\zf@update@family</code> 125, <u>327</u> , 393, 407, 555, 577, 582, 598, 610, 630, 634, 637, 666, 670, 673, 676, 681, 684, 836, 1023
<code>\zf@update@ff</code> . . .	126, <u>375</u> , 394, 408, 635, 667, 671, 674, 677, 685, 837, 1024
<code>\zf@v@strnum</code>	<u>452</u> , 494
<code>\zf@wordspace@parse</code>	612, <u>614</u>