

## Part I

# Commands for accents and symbols (‘encodings’)

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

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

Multilingual documents will often use different fonts for different languages, not just for style, but for the more pragmatic reason that fonts do not all contain the same glyphs. (In fact, only test fonts such as Code2000 provide anywhere near the full Unicode coverage.) Indeed, certain fonts may be perfect for a certain application but miss a handful of necessary diacritics or accented letters. In these cases, `fontspec` can leverage the font encoding technology built into  $\text{\LaTeX} 2_{\epsilon}$  to provide on a per-font basis either provide fallback options or error messages when a desired accent or symbol is not available. However, these features can only be provided for input using  $\text{\LaTeX}$  commands rather than Unicode input; for example, typing `\`e` instead of `è` or `\textcopyright` instead of `©` in the source file.

The most widely-used encoding in  $\text{\LaTeX} 2_{\epsilon}$  was T1 with companion ‘TS1’ symbols provided by the `textcomp` package. As of 2017, in  $\text{\LaTeX} 2_{\epsilon}$  on  $\text{\XeTeX}$  and  $\text{\LuaTeX}$ , the default encoding is known as TU, which emulates the basic coverage of T1+TS1 for command-based input of accents and symbols. Wider coverage is not provided by default since it is expected that most users will be writing with direct Unicode input.

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

## 1 A new Unicode-based encoding from scratch

Let’s say you need to provide backwards-compatible support for the OT2 encoding, which contains definition for Cyrillic letters. An example from the OT2 encoding definition file (`ot2enc.def`) reads:

```
57 \DeclareTextSymbol{\CYRIE}{OT2}{5}  
58 \DeclareTextSymbol{\CYRDJE}{OT2}{6}  
59 \DeclareTextSymbol{\CYRTSHE}{OT2}{7}  
60 \DeclareTextSymbol{\cyrnje}{OT2}{8}  
61 \DeclareTextSymbol{\cyrlje}{OT2}{9}  
62 \DeclareTextSymbol{\cyrdzhe}{OT2}{10}
```

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

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

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

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

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

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

Then follow it up with a font loading call such as

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

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

## 2 Adjusting a pre-existing encoding

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

When adding symbols, etc., simply write

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

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

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

```

\DeclareUnicodeEncoding{nobang}{
  \ImportEncodingFile{tuenc.def}
  \UndeclareSymbol\textinterrobang
}

```

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

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

```

\EncodingAccent{\d}{"0323}

```

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

```

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

```

This would be set up in a document as such:

```

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

```

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

```

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

```

To reiterate from above, typing this input with Unicode text (‘kalita<sub>m</sub>’) will bypass this encoding mechanism and you will receive only what is contained literally within the font.

### 3 Summary of commands

The  $\LaTeX 2_{\epsilon}$  kernel provides the following font encoding commands suitable for Unicode encodings:

```

\DeclareTextCommand{<command>}{<encoding>}[<num>][<default>]{<code>}
\DeclareUnicodeAccent{<command>}{<encoding>}{<slot>}
\DeclareTextSymbol{<command>}{<encoding>}{<slot>}

```

```

\DeclareTextComposite{<command>}{<encoding>}{<letter>}{<slot>}
\DeclareTextCompositeCommand{<command>}{<encoding>}{<letter>}{<code>}
\UndeclareTextCommand{<command>}{<encoding>}

```

See [fntguide.pdf](#) for full documentation of these. As shown above, the following shorthands are provided by fontspec to simplify the process of defining Unicode font range encodings:

```

\ImportEncodingFile{<filename>}
\EncodingCommand{<command>}[<num>][<default>]{<code>}
\EncodingAccent{<command>}{<code>}
\EncodingSymbol{<command>}{<code>}
\EncodingComposite{<command>}{<letter>}{<slot>}
\EncodingCompositeCommand{<command>}{<letter>}{<code>}
\UndeclareSymbol{<command>}
\UndeclareComposite{<command>}{<letter>}

```

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