Putting the Cork back in the bottle — Improving Unicode support in TEX

Mojca Miklavec
Faculty of Mathematics and Physics, University of Ljubljana

Arthur Reutenauer
GUTenberg, France
http://tug.org/tex-hyphen

Abstract

Until recently, all of the hyphenation patterns available for different languages in TEX were using 8-bit font encodings, and were therefore not directly usable with UTF-8 TEX engines such as XETEX and LuaTEX. When the former was included in TEX Live in 2007, Jonathan Kew, its author, devised a temporary way to use them with XETEX as well as the "old" TEX engines. Last spring, we undertook to convert them to UTF-8, and make them usable with both sorts of TEX engines, thus staying backwardly compatible. The process uncovered a lot of idiosyncrasies in the pattern-loading mechanism for different languages, and we had to invent solutions to work around each of them.

1 Introduction

Hyphenation is one of the most prominent features of TEX, and since it is possible to adapt it to many languages and writing systems, it should come as no surprise that there were so many patterns created so quickly for so many languages in the relatively early days of TEX development. As a result, the files that are available often use old and dirty tricks, in order to be usable with very old versions of TEX. In particular, all of them used either 8-bit encodings or accent macros (`\'e`, `\v{z}`, etc.); Unicode did not yet exist when most of these files were written.

This was a problem when XETEX was included in TEX Live in 2007, since it expects UTF-8 input by default. Jonathan Kew, the XETEX author, devised a way of using the historical hyphenation patterns with both XETEX and the older extensions of TEX: for each pattern file ⟨hyph⟩`.tex`, he wrote a file called `xu-`⟨hyph⟩`.tex` that detects if it is run with XETEX or not; in the latter case, it simply inputs ⟨hyph⟩`.tex` directly, and otherwise, it takes actions to convert all the non-ASCII characters to UTF-8, and then inputs the pattern file.

To sum up, in TEX Live 2007, XETEX used the original patterns as the basis, and converted them to UTF-8 on the fly.

In the ConTEXt world, on the other hand, the patterns had been converted to UTF-8 for a couple of years, and were converted back to 8-bit encodings by the macro package, depending on the font encoding.

In an attempt to go beyond that and to unify those approaches, we then decided to take over conversions for all the pattern files present in TEX Live at that time (May 2008), for inclusion in the 2008 TEX Live release.

2 The new architecture

The core idea is that after converting the patterns to UTF-8, the patterns are embedded in a structure that can make them loadable with both sorts of TEX engines, the ones with native UTF-8 support (XETEX, LuaTEX) as well as the ones that support only 8-bit input.[1]

The strategy for doing so was the following: for each language ⟨lang⟩, the patterns are stored in a file called `hyph-`⟨lang⟩`.tex`. These files contain only the raw patterns, hyphenation exceptions, and comments. They are input by files called `loadhyph-`⟨lang⟩`.tex`. This is where engine detection happens, such as this code for Slovenian:

```
% Test whether we received one or two arguments
\def\testengine#1#2!{\def\secondarg{#2}}
% We are passed Tau (as in Taco or TEX,
% Tau-Epsilon-Chi), a 2-byte UTF-8 character
\testengine T!\relax
% Unicode-aware engines (such as XeTeX or LuaTeX)
% only see a single (2-byte) argument
\ifx\secondarg\empty
\message{UTF-8 Slovenian Hyphenation Patterns}
\else
\message{EC Slovenian Hyphenation Patterns}
\input conv-utf8-ec.tex
\fi
\input hyph-sl.tex
```

The only trick is to make TEX look at the Unicode character for the Greek capital Tau, in UTF-8 encoding: it uses two bytes, which are therefore read by 8-bit TEX engines as two different characters; thus

[1] A note on vocabulary: in this article, we use the word "engine" or "TEX engine" for extensions to the program TEX, in contrast to macro packages. We then refer to (TEX) engines with native UTF-8 support as "UTF-8 engines", and to the others as "8-bit engines", or sometimes "legacy engines", borrowing from Unicode lingo.

the macro `\testengine` sees two arguments. UTF-8 engines, on the other hand, see a single character (Greek capital Tau), thus a single argument before the exclamation mark, and `\secondarg` is `\empty`.

If we're running a UTF-8 TEX engine, there is nothing to do but to input the file with the UTF-8 patterns; but if we're running an 8-bit engine, we have to convert the UTF-8 byte sequences to a single byte in the appropriate encoding. For Slovenian, as for most European languages written in the Latin alphabet, it happens to be `T1`. This conversion is taken care of by a file named `conv-utf8-ec.tex` in our scheme. Let's show how it works with these three characters:[2]

'č' (`UTF-8` ⟨`0xc4, 0x8d`⟩, `T1 0xa3`),

'š' (`UTF-8` ⟨`0xc5, 0xa1`⟩, `T1 0xb2`),

'ž' (`UTF-8` ⟨`0xc5, 0xbe`⟩, `T1 0xba`).

In order to convert the sequence ⟨`0xc4, 0x8d`⟩ to `0xa3`, we make the byte `0xc4` active, and define it to output `0xa3` if its argument is `0x8d`.[3] The other sequences work in the same way, and the extracted content of `conv-utf8-ec.tex` is thus:[4]

```
\catcode"C4=\active
\catcode"C5=\active
%
\def^^c4#1{%
\ifx#1^^8d^^a3\else % U+010D
\fi}
%
\def^^c5#1{%
\ifx#1^^a1^^b2\else % U+0161
\ifx#1^^be^^ba\else % U+017E
\fi\fi}
% ensure all the chars above have valid lccode's:
\lccode"A3="A3 % U+010D
\lccode"B2="B2 % U+0161
\lccode"BA="BA % U+017E
```

As the last comment says, we also need to set non-zero `\lccode`s for the characters appearing in the pattern files, a task formerly carried out in the pattern file itself.

The information for converting from UTF-8 to the different font encodings has been retrieved from the encoding definition files for LaTeX and ConTEXt, and gathered in files called ⟨`enc`⟩`.dat`. The converter files are automatically generated with a Ruby script from that data.

[2] The only non-ASCII characters in Slovenian.

[3] The same method would work flawlessly if the sequence contained three or more bytes — although this case doesn't arise in our patterns — since the number of bytes in a UTF-8 sequence depends only on the value of the first byte.

[4] Problems would happen if a `T1` byte had been made active in that process, but for reasons inherent to the history of TEX font encodings, as well as Unicode, this *is never the case for the characters used in the patterns*, a fact the authors consider a small miracle. The proof of this is much too long to be given in this footnote, and is left to the reader.

Here is a table of the encodings we support:

ConTEXt	LaTeX	Comments
`ec`	`T1`	"Cork" encoding
`il2`	`latin2`	ISO 8859-2
`il3`	`latin3`	ISO 8859-3
`lmc`	`lmc`	`montex` (Mongolian)
`qx`	`qx`	Polish
`t2a`	`t2a`	Cyrillic

2.1 Language tags: BCP 47 / RFC 4646

A word needs to be said about the language tags we used. As a corollary to the completely new naming scheme for the pattern files and the files surrounding them, we wanted to adopt a consistent naming policy for the languages, abandoning the original names completely, because they were problematic in some places. Indeed, they used ad hoc names which had been chosen by very different people over many years, without any attempt to be systematic; this has led to awkward situations; for example, the name `ukhyphen.tex` for the British English patterns: while "UK" is easily recognized as the abbreviation for "United Kingdom", it could also be the abbreviation for "Ukrainian", and unless one knows all the names of the pattern files by heart, it is not possible to determine what language is covered by that file from the name alone.

It was therefore clear that in order to name files that had to do with different *languages*, we had to use language codes, not country codes. But this was not sufficient either, as can be seen from the example of British English, since it's not a different language from English.

Upon investigation, it turned out that the only standard able to distinguish all the patterns we had was the IETF "Best Current Practice" recommendation 47 (BCP 47), which is published as RFC documents; currently, it's RFC 4646.[5] This addresses all the language variants we needed to tag:

- Languages with variants across countries or regions, like English.
- Languages written in different scripts, like Serbian (Latin and Cyrillic).
- Languages with different spelling conventions, like Modern Greek (which underwent a reform known as *monotonic* in 1982), and German (for which a reform is currently happening, started in 1996).

[5] In the past, it has been RFC 1766, then RFC 3066, and is currently being rewritten, with the working title RFC 4646bis. RFC 4646 is available at `ftp://ftp.rfc-editor.org/in-notes/rfc4646.txt`, and the current working version of RFC 4646bis (draft 17) at `http://www.ietf.org/internet-drafts/draft-ietf-ltru-4646bis-17.txt`.

A list of all the languages with their tags can be found in appendix A.

3 Dealing with the special cases

There were so many special cases that one might say that the generic case was the special one!

3.1 Pattern files designed for multiple encodings

The first problem we encountered was with patterns that tried to accommodate both the OT1 and the T1 encoding in the same file.

The first language for which this had been done was, historically, German, and the same scheme was subsequently adopted for French, Danish, and Latin. The idea is the following: in each of these languages, there are characters that are encoded at different positions in OT1 and in T1; for German, it is the sharp s 'ß'; for French, it is the character 'œ', etc. In order to deal with that, each pattern that happened to contain one of these characters was duplicated in the file, with intricate macros to ignore them selectively, depending on the font encoding used.

This would have been very awkward to reproduce in our architecture, if at all possible: it would have meant that each word such as, say, "cœur" in French would need to yield two different byte strings in 8-bit mode, for OT1 and T1 (c^^1bur and c^^f7ur, respectively). We therefore decided to put the duplicate patterns in a separate file called spechyph-⟨lang⟩-ot1.tex that is input only in legacy mode, after the main file hyph-⟨lang⟩.tex.

The patterns packaged in this fashion should therefore behave in the same way as the historical files, enabling a few breakpoints with non-ASCII characters in OT1 encoding. We would like to stress, though, that OT1 is definitely not the way to go for these languages. We only supported this behaviour for the sake of compatibility, but we doubt it is very useful: if one uses OT1 for German or French, one would indeed have a few patterns with 'ß' or 'œ', respectively, but many more patterns, with accented characters, would be missed. In order to take full advantage of the hyphenation patterns, one needs to use T1 fonts.

It has to be noted that in addition, we ended up not using the aforementioned approach in the case of German, because we wanted to account for the ongoing work to improve the German patterns; thus, we decided to use the new patterns with the UTF-8 engines, but not with the 8-bit engines, for compatibility reasons. In the latter case, we simply include the original pattern file in T1 directly, with no conversion whatsoever. For the three other languages, though (French, Danish and Latin), we used a spechyph-⟨lang⟩-ot1.tex file.

3.2 Multiple pattern sets for the same language

Another interesting issue was with Ukrainian and Russian, where different complications arose.

First, the pattern files were also devised for multiple encodings, but in a different manner: here, the encoding is selected by setting the control sequence \Encoding before the pattern file is loaded. Depending on the value of that macro, the appropriate conversion file is then input, that works in the same way as our conv-utf8-⟨enc⟩.tex files. There is of course a default value for \Encoding, which for both languages is T2A,[6] the most widespread font encoding for Russian and Ukrainian, and the one used in the pattern files; thus, no conversion is necessary if \Encoding is kept to its default value.

Then, both Russian and Ukrainian had several pattern files, with different authors and/or hyphenation rules (phonetic, etymological, etc.). Those were selected with a control sequence called \Pattern, by default as for Russian (by Aleksandr Lebedev), and mp for Ukrainian (by Maksym Polyakov).

Both those choices could, of course, be overridden only at format-building time, since the patterns are frozen at that moment.

Finally, they used a special trick, implemented in file hypht2.tex, to enable hyphenation inside words containing hyphens, similar to Bernd Raichle's hypht1.tex for T1 fonts.

Those three features had to be addressed in very different ways in our structure: while the first one was irrelevant in UTF-8 mode, it would have implied fundamental changes in our loadhyph-⟨lang⟩.tex files for 8-bit engines, since the implicit assumption that any language uses exactly one 8-bit encoding would no longer be met. The second feature was easier to handle, but still demanded additional features in our loadhyph-⟨lang⟩.tex files. Finally, the third feature, although certainly very interesting, seemed more fragile than what we felt was acceptable.

Upon deliberation, we then decided to not include those features in the UTF-8 patterns before TEX Live 2008 was out, but to still enable them in legacy mode, in order to ensure backward compatibility. And thanks to subsequent discussions with Vladimir Volovich, who devised the way the Russian patterns were packaged, and inspired the Ukrainian ones, we could include a list of hyphenated compound words which we put in files called exhyph-ru.tex

[6] Actually t2a, lowercase.

and `exhyph-uk.tex`, respectively. The strategy we used is thus:

In UTF-8 mode, input the UTF-8 patterns, then the `ex-` file.

In legacy mode, simply input the original pattern file directly.

Therefore, the only feature missing, overall, in TEX Live 2008, is the ability to choose one's favorite patterns in UTF-8 mode: for each language, we only converted the default set of patterns to UTF-8. Setting `\Pattern` will thus have no effect in this case, but it will behave as before in 8-bit mode. Now that TEX Live 2008 has been released we intend to change that behaviour soon, and to enable the full range of features that the original pattern files had.

It should also be noted that in TEX Live 2007, Bulgarian used the same pattern-loading mechanism, but that there was actually only one possible encoding, and only one pattern file, so there was no real choice, and it was therefore straightforward to adapt the Bulgarian patterns to our new architecture.

4 TEX Live 2008

The result of our work has been put on CTAN under the package name hyph-utf8, and is the basis for hyphenation support in TEX Live 2008. We don't consider our work to be finished (see next section), and we welcome any discussion on our mailing-list (`tex-hyphen@tug.org`). We also have a home page at `http://tug.org/tex-hyphen`, to which readers are referred for more information.

The package has been released in the TDS layout, with the TEX files in `tex/generic/hyph-utf8` and subdirectories. The encoding data and Ruby scripts are available in `source/generic/hyph-utf8`. Some language-specific documentation has been put in `doc/generic/hyph-utf8`.

5 And now ...

There still are tasks we would like to carry out: the `hypht1.tex` / `hypht2.tex` behaviour has already been mentioned, and one of the authors has lots of ideas on how to improve Unicode support *yet more* in UTF-8 TEX engines.

We appeal to pattern authors to make contact with us in order to improve and enhance our package; many of them have already communicated with us, to our greatest pleasure, and we're confident that our effort will be understood by all the developers dealing with language-related problems.[7]

Among the immediate and practical problems is, in particular:

5.1 ... for something completely different

Babel would need to be enhanced in order to enable different "variants" for at least two languages. One is Norwegian, for which two written forms exist, known as "bokmål" and "nynorsk" (ISO 639-1 `nb` and `nn`, respectively).[8] At the moment, Babel has only one "Norwegian" language. The second is Serbian, which can be written in both the Latin and the Cyrillic alphabets; these possible variants which are not yet taken into account in Babel.

6 Acknowledgements

First and foremost, we wish to thank wholeheartedly Karl Berry, who supported the project from the beginning and guided us with advice, as well as Hans Hagen, Taco Hoekwater and Jonathan Kew, for their technical help, and, finally, Norbert Preining, who went through the trouble of integrating the new package into TEX Live.

Appendix A List of supported languages

`ar`	Arabic	`la`	Latin
`fa`	Farsi	`mn-cyrl`	Mongolian
`eu`	Basque	`mn-cyrl-x-2a`	Mongolian (new patterns)
`bg`	Bulgarian	`no`	Norwegian
`cop`	Coptic	`nb`	Norwegian Bokmål
`hr`	Croatian	`nn`	Norwegian Nynorsk
`cs`	Czech	`zh-latn`	Chinese Pinyin
`da`	Danish	`pl`	Polish
`nl`	Dutch	`pt`	Portuguese
`eo`	Esperanto	`ro`	Romanian
`et`	Estonian	`ru`	Russian
`fi`	Finnish	`sr-latn`	Serbian, Latin script
`fr`	French	`sr-cyrl`	Serbian, Cyrillic script
`de-1901`	German, "old" spelling	`sh-latn`	Serbo-Croatian, Latin script
`de-1996`	German, "new" spelling	`sh-cyrl`	Serbo-Croatian, Cyrillic script
`el-monoton`	Monotonic Greek	`sl`	Slovene
`el-polyton`	Polytonic Greek	`es`	Spanish
`grc`	Ancient Greek	`sv`	Swedish
`grc-x-ibycus`	Ancient Greek, Ibycus encoding	`tr`	Turkish
`hu`	Hungarian	`en-gb`	British English
`is`	Icelandic	`en-us`	American English
`id`	Indonesian	`uk`	Ukrainian
`ia`	Interlingua	`hsb`	Upper Sorbian
`ga`	Irish	`cy`	Welsh
`it`	Italian		

[7] The acknowledgement section, had it been as long as the authors would have wished it to be, would have more than doubled the size of this article.

[8] The ISO standard also includes a code for "Norwegian", `no`, although this name is formally ambiguous.