Surviving in a multi-lingual world with multiple font encodings

Jiří Zlatuška
Faculty of Informatics
Masaryk University
Burešova 20
602 00 Brno, Bohemia
zlatuska@informatics.muni.cz

Abstract

This paper develops constructions needed for utilizing the encoding scheme concept embedded into LaTeX 2$_\varepsilon$ in order to develop system allowing to use different font encoding schemes and different languages within one format and to provide mechanisms for fully functional switching between them. The concept of languge extends that of TeX based just on hyphenation patterns. A practical demostration of this is shown on the example defining hyphen-splitting modification to standard line-breaking behaviour of TeX.

Introduction

In the new standard LaTeXversion, LaTeX 2$_\varepsilon$[3, 2, 4], there is a provision for living with more versatile font organizations than those assumed by Donald Knuth in the TeXbook. This version of the system has incorporated the so-called *New Font Selection Scheme version 2* (NFSS2) as an integral part of LaTeX 2$_\varepsilon$ kernel allowing not only to overcome some deficiencies of the original LaTeX as far as switching between various font styles, but also to incorporate handling of fonts with layouts via the concept of *font encoding scheme*.

The introduction of font encoding schemes made it possible for incorporation of fonts organized in different ways into a single format. Standard LaTeX 2$_\varepsilon$ makes use of it by including both the original font organization used by Knuth (within LaTeX 2$_\varepsilon$ called the *old* TeX scheme, OT*n*) and the the so-called *Cork scheme* (denoted T1 as for extended TeX text scheme within LaTeX 2$_\varepsilon$) designed specifically to suit most of the European languages. From the viewpoint of creating a new standard for portable texts, both of the encodings are important to be present in the new system: the former provides for backward compatibility with the world of TeX fonts as used until very recently, while the latter is a part of a complete set of fonts prepared for 8-bit TeX (the font organization for 8-bit Math fonts accompanying the text encoding T1 is yet to come).

Font encoding T1 provides a layout sufficient for typesetting texts in all European languages based on the Latin alphabet. Even though it is desirable to use this "universal" font encoding scheme to ensure portability of the source document texts, for practical reasons of continuity within local groups or installations it is often desirable to allow alternative use of different encodings so that backward compatibility is ensured as well as the use of local fonts with possibly different layouts. The concept of encoding scheme can be used in order to allow for this flexibility, however some problems emerge when this functionality needs to be properly coupled with the mechanisms for switching the basic language-dependent features such as the hyphenation settings. Some of these problems will be discussed in the present paper along with a proposal for the low-level interface for hyphenation settings definitions in the presence of possibly several different font encodings, and for the functionality needed in order to ensure transparent behaviour of both font encoding and hyphenation switching. It is assumed that this low-level interface can be incorporated into packages which aim at formation of a complete language-specific environment such as language-specific style files or an integrated system providing several of them in a uniform way such as, e.g., the *Babel* system [1].

Font encodings

The concept of font encoding presents a consistent way of introducing definitions local to a particular font layout and of ensuring that the proper definition is used within the currently selected font layout.

Within LaTeX 2$_\varepsilon$, font encodings present a parameter to every font-specific definition. The list of these definitions include commands specific to a particular encoding. The typical location of these definitions for encoding `XXX` is the encoding definition file `XXXenc.def` which is loaded by LaTeX 2ε when the `\DeclareFontEncoding` command is processed. Font encoding definition files contain most but not all of the definitions and settings necessary for proper function of a particular font layout within TeX. Among those missing are definitions of the `\catcode`, `\lccode`, `\uccode` and `\sfcode` values for a particular font layout. These are only defined once and for all within the LaTeX 2ε kernel—obviously in order to avoid the overhead of 4×256 redefinitions occurring each time a font with different font encoding is selected (also note that this number exceeds the "standard" number of the *save stack* entries of 600 assumed by most of the common TeX implementations).

The use of fonts populated by up to 256 characters brings around the possibility of introducing language-specific hyphenation patterns based on a particular font layout. LaTeX 2ε makes this even more accessible by redefining the accent commands of `plain` such as, e.g., `\v`, `\'`, etc., by definitions which do not use the `\accent` primitive of TeX, but which instead generate the corresponding composite character as present directly in the font (if there is not any, such as in the `OT1` standard TeX text encoding). This provides an encoding-independent way (7-bit) of representing the accented letters, yet without sacrificing features which depend on the proper representation (8-bit) such as, e.g., the proper use of the hyphenation patterns—captions in the style files can be written in this way without committing to any particular encoding under which the style file can possibly be used.

One of the consequences of the introduction of accent commands which evaluate directly into the accented letter included in the font is the possibility of creating *encoding-independent hyphenation patterns definitions.* TeX expands macros encountered during `initex` input of the `patterns` command. Therefore if only expandable tokens and terminal letters are generated (i.e. no executable commands such as `\def`s or `\accent`s), encoding-independent hyphenation patterns can be processed correctly with the result being dependent on the actual definition of the accent commands within a particular font encoding scheme.

Working with multiple encodings

When working with multiple encodings, one has to ensure that all of the parameters of TeX's work which depend on the encoding selected are changed properly when switching from one encoding into another.

Clearly, there is little problem in doing so as far as encoding-dependent definitions are concerned. Here LaTeX 2ε already provides all what is needed: When defining an encoding-specific command `\com` for encoding `XXX` (e.g., an accent command or a command generating a glyph from a particular position from `XXX`-encoded font), LaTeX 2ε actually defines two macros: one of them serving just for the internal purposes, `\XXX\com`, and the other one being the user-level macro `\com`.

The internal macro `\XXX\com` takes as its definition the code supplied by the user as code defining `\com` in the encoding `XXX`.

The user-accessible macro `\com` which takes the currently active encoding scheme, `YYY`, and then proceeds by invoking internal command `\YYY\com` of by issuing an error message if the latter is not defined. The actual definition is slightly more complicated because `\com` is actually defined as `protect`ed macro and for the reasons of efficiency one additional internal macro is used which prevents repeated construction of `\YYY\com` using `\csname`...`\endcsname` when using the same command under unchanged font encoding scheme.

A serious problem arises with switching encoding definitions while requiring at the same time that hyphenation works correctly within each of those encodings. Here TeX allows only a partial solution to this problem. When building paragraphs, TeXonly works with the definitions of the values assigned to `\lccode` at the moment when the end of the paragraph is reached. Yet, the correct assignment of `\lccode` to each letter within the paragraph to be broken is essential for correct work of TeX's hyphenation algorithm as far as each word tried for hyphenation points is first converted into its lower-case version. Therefore no matter that TeX 3 is capable of inserting the correct `\language` nodes within the paragraph ensuring proper switching of hyphenation patterns, they will only be processed correctly when no changes are needed to the `\lccode` values for any of the encodings which are used within a paragraph.

This feature built into TeX the program limits the practical usability of heavy switching between incompatible encoding schemes for the input text. LaTeX 2ε actually places the definitions of the `\lccode`, `\uccode`, and `\sfcode` arrays into the kernel and makes them fixed once the LaTeX 2ε format is generated. Clearly, the underlying assumption was that `T1` is the encoding of choice and the text to be processed either honours these definitions, or just changes them once without the need of returning back to them (e.g., for using local packages which introduce different font encoding conventions). When working in an installation which needs to support different and incompatible font encodings (e.g., for historical reasons or local backward compatibility), one needs to introduce these definitions as a part of the `\DeclareFontEncoding` definition of the encoding included in `fontdef.cfg` installation-dependent file used for format generation.

Encoding-specific hyphenation patterns

Although it is possible to provide accent command definitions which allow the argument to the `\patterns` operation to be presented in an encoding-independent way, the actual way how patterns are stored within TEX and subsequently processed into tries used for locating hyphenation points does depend on the encoding definition—the `\lccode` values are essential to this in the same way as during actual paragraph breaking and of course the codes of accented letters generated as the result of expansion of the accent commands do matter as well.

In order to be able to use the same set of hyphenation patterns within several font encodings, it is therefore necessary to (re-)load a particular set of hyphenation patterns as a separate "language" (in the terminology of TEX when introducing the `\language` primitive) versions of the same set. This means the number assigned to the particular set of patterns loaded no longer corresponds alone to the language–it is the language *and* the corresponding encoding instead.

Language-switching requirements

As we stated earlier, we are only interested in considering the requirements of the low-level language switching, i.e. language switching ensuring the text gets properly formatted, but not caring about the actual texts an/or other language-related higher-level entities of the document formatting environment. What is the interface needed for this low-level language switching?

First, a command for switching into a particular language is needed, yet it cannot be safely assumed that this could correspond to a particular `\language` if the possibility of transparent use of several font encodings is to be ensured. If we take the Czech language as an example, then the language-switching command should be, e.g.,

```
\czech
```

or (as in `babel`)

```
\selectlanguage{\czech}
```

but the possibility which should outright be excluded is `\language=`<*number*> as far as this exposes too much of a detail of the internal implementation and it is clearly impossible to ensure that the same `\language` number can be used in every installation no matter which local font encodings are used.

Although TEX treats a "language" as merely a set of hyphenation patterns, it is more natural also to include setting of other parameters as a part of the language-switching action, most notably the `\lefthyphenmin` and `\righthyphenmin` values (hyphenation patterns are usually tuned to some particular values of these). Non-standard features associated with hyphenation in certain languages also seem natural to be added at this level—we show an example of this later in this paper.

Second, when changing font encoding the hyphenation patterns need to be switched automatically in synchronization with this so that when the user selects Czech hyphenation with internal `\language` identifier equal to 2 within `T1` encoding and than issues `\fontencoding{IL2}` in order to switch to ISO Latin 2 encoding scheme, a side effect of this should be change of the current `language` setting to appropriate value, e.g., 3. This is not the same as switching to a new font encoding and then issuing the command for switching into the current language. The user might have changed some of the parameters associated by default with the language (e.g., the `left/right-hyphenmin` values) and the system should honour this change—just the `\language` should be updated, nothing more.

Third, during `initex` processing definition of new languages available within the system should be tied to reading the hyphenation patterns in. Definitions of the hyphenation patterns should be re-read for each relevant encoding and the appropriate additional actions should be associated with the set of hyphenation patterns on the level of language "declaration".

Setting up the internals

LATEX 2_ε has set up quite a high standard of portability of the source documents and excluded much of the possibilities for nonstandard modifications of the system setup during generation of the format file. Hence all the extensions can only be made within the configuration files allowed by the new standard.

The principal place where essentially all what is needed can be performed is the file used for configuring the system to a particular set of hyphenation patterns, i.e. `lthyphen.cfg`. All of the following code can be put in there. The definitions of language-switching commands can be linked to loading the corresponding hyphenation patterns in. There will be two kinds of them—one for encoding-independent languages (e.g., English in most common settings), and another one for languages where the change of font encoding scheme implies change of the `\language` parameter as well. The former kind corresponds to defining the switching instructions using `\newcommand`, and the latter to using `\DeclareTextCommand` for making the definition encoding-specific.

The basic command providing the setup of hyphenation patterns and generating the definitions of the corresponding language-switching commands now has the format

```
\DeclareLanguage[<cdp>*]
                {<file>}
                {<\cmd>}
                {<...env...>}
```

where <*cdp*>* is a comma-separated list specifying the encodings for which to ensure an encoding-specific setup using the file <*file*> containing the hyphenation patterns and hyphenation exceptions for a particular language for which the language-switching command \\<*cmd*> with additional setup determined by the piece of code <...*env*...> should be generated.

When invoking this command with empty font-encoding list, the resulting definition of \\<*cmd*> is obtained simply by direct use of `\def`. When <*cdp*>* is non-empty, then for each encoding from this list the encoding-specific language switching command is generated by using construction effectively equivalent to

```
\DeclareTextCommand{<cdp>}
          {\<cmd>}
          {\language=n
           \let\cur@lang=\<cmd>
           <...env...>}
```

where n stands for the `\language` number under which the patterns and exceptions have been loaded. For language which do not depend on font encodings (that is they work with any of the font encodings used within the installation) it is sufficient use ordinary `\def` command, that is

```
\newcommand{\<cmd>}
          {\language=n
           \let\cur@lang=\<cmd>
           <...env...>}
```

These definitions provide for the user-level selection of the language features. As noted above, another version is needed for internal hyphenation patterns switching without affecting the other parameters which might meanwhile have been changed by the user. These definitions can be defined using just versions of the user-level names, i.e.

```
\DeclareTextCommand{<cdp>}
                   {\@<cmd>}
                   {\language=n}
```

and

```
\newcommand{\@<cmd>}
           {\language=n}
```

The use of these internal switching commands has to be linked to the font-encoding switching process. This can most conveniently be done by including a `\@hyphenation` command into the text settings of the `DeclareFontEncoding` command (usually found either in `fontdef.cfg` and/or the encoding definition files <*ENC*>`enc.def` files) for every encoding definition for which encoding-specific hyphenation may be loaded. The definition of this synchronization command just looks on the current language activated and invokes the corresponding internal command:

```
\def\@hyphenation{\csname
         @\expandafter\@gobble
         \string\cur@lang
                 \endcsname
}
```

The code outlined above works fine provided that the encodings used share the `\catcode` and `\lccode` definitions. The code resetting these can be added into `\@hyphenation` so that the tables defining the characteristics of the setup are updated each time encoding scheme is switched. As noted earlier in this paper, this does not make the behaviour absolutely perfect: It will only work for full paragraphs because of the way TEX uses the associated information during paragraph breaking. If used together with delimiting the scope of language/encoding specification, it will put an extremely heavy burden on the save stack—more than what compile-time parameters of common TEX implementations are prepared to withstand.

Loading encoding-independent hyphenation patterns

Within LaTeX 2$_\varepsilon$'s way of optimizing the code of encoding-specific commands, it is not possible to include accent commands defined using `\DeclareTextComposite` directly. First, they are `\protect`ed, and this leads to the `\relax` command. Second, they may use `\def` command internally. Either of these lead to an executable command which is not allowed in the expand-only mode used for `\patterns` processing.

This can be solved by redefining temporarily certain commands invoked internally during accent commands processing so that they worked purely by expansion. In the phase of reading in the hyphenation patterns and exceptions this can be done locally within a group because both `\patterns` and `\hyphenation` always work globally. Certain care must be taken however if the language-switching commands are to be generated at the same time as far as there isn't any `\global` prefix for either `\newcommand` or `\DeclareTextCommand` instructions and thus their actual implementation will look slightly more verbose compared with what we shown in the preceding section.

The redefinitions needed for inputting hyphenation patterns prepared using the accent command are shown in Figure 1 (this code is mostly due to Alan Jeffery and it works with `<1994/06/01>` release of LaTeX 2$_\varepsilon$). Note that the code includes two possibilities for an error condition discovered and it tries to fire the standard mechanism for reporting errors. When used within `\patterns`, this will work because this would require also execution, not just expansion. Hence iniTeX reports "bad patterns" if issuing error message is attempted.

Special language-dependent setup

For proper language-dependent formatting just hyphenation patterns are clearly not enough. When introducing the `\DeclareLanguage` command, we have included also a general piece of code as a part of the setup. In most cases this is just reserved for the `\lefthyphenmin` and `\righthyphenmin` parameters. In certain situations this can be however used for implementation of additional features linked with hyphenation but but not included in the TeX code.

As an example of a more complicated feature of this kind, let us consider a modified behaviour of the hyphenation process when single dash is encountered within a word. The proper typesetting practice in Czech calls for placing the hyphen twice if hyphenating at this point: the first one at the end of one line, and the other at the beginning of the second line.

The code implementing this hyphen-splitting property is shown on Figure 2 (part of the code was suggested by Phil Taylor). Note that because hyphen characters are used within TeX also for inputting negative numbers, the active version of '-' must first try to narrow the modes in which this feature is meaningfuly applicable. When any other mode except the unrestricted horizontal mode is found, a token corresponding to the ordinary hyphen character is returned. When in the horizontal mode, the code looks ahead and tries to emulate the usual ligatures found in Computer Modern typefaces by checking for numbers of subsequent hyphens and then re-issuing the hyphen characters so that the ligatures are built by the ligature building mechanism in order to have this process as much font-independent as possible. When single hyphen is found, before issuing the `\discretionary` a check is made for subsequent 'l': there is a common suffix '-li' used in Czech before which the compound word cannot be split unless `\righthyphenmin`<3.

When attaching this feature to the language-specific code within the `\DeclareLanguage` command, it has to be activated and de-activated upon request, so that certain languages can switch this off even thow the language from where they have been called makes active use of it. The corresponding set of definitions may be used in order to do so:

```
\newif\if@splithyphens
\newif\if@splitrequested
\def\standardhyphens{\catcode`-=12
                 \@splithyphensfalse}
\def\splithyphens{\if@splitrequested
                    \catcode`-\active
                 \fi
                 \@splithyphenstrue}
\def\@requesthyphens{%
                 \if@splithyphens
                    \catcode`-=\active
                 \fi
                 \@splitrequestedtrue}
```

Two conditions are defined here: The first of them, `@splitrequested` is satisfied when the hyphen-splitting feature has been activated from the user level—usually using the `\splithyphens` command no matter whether the language actually allowed it to become active *or* when the language activated allows the hyphen-splitting code to be activated. The second one, `splithyphens`, becomes true when the

```
\bgroup
   \lccode`\|=`\\
   \lowercase{
\egroup
   \def\@changed@x@exp|#1|#2\@changed@x{%
      \expandafter\ifx\csname\f@encoding |#2\endcsname\relax
         \@latexerr{Command |#2 undefined in encoding \f@encoding}\@eha
      \else
         \csname\f@encoding |#2\expandafter\endcsname
      \fi}
  }
\def\@text@composite@chk#1#2{%
   \expandafter\ifx\csname\string#1-\string#2\endcsname\relax
         \@latexerr{Undefined accent-character combination.}\@eha
   \else
         \csname\string#1-\string#2\expandafter\endcsname\expandafter
         \@firstofone
   \fi}
\def\expand@accents{%
   \let\protect\@empty
   \let\@changed@x\@changed@x@exp
   \let\@text@composite\text@composite@chk
   \expandafter\let\csname\f@encoding-cmd\endcsname\@empty}
```

Figure 1: Code needed for expand-only processing of LaTeX 2_ε accent commands.

hyphen-splitting has been acivated by the user (usually using the `\splithyphens` for switching on and `\standardhyphens` for switching off). Languages in which hyphen-splitting is not allowed simply execute `@splitrequestedfalse` as a part of their <…*env*…> code specified in the last argument of `\DeclareLanguage`. Languages, for which this feature can be activated, execute `\@requesthyphens` which either turns hyphen-switching on (when `splithyphens` has already been issued), or will make subsequent `splithyphens` turning it on immediately.

One of the reasons for user-selectable hyphen splitting code is that that the code for active hyphen is not entirely fool-proof. It cannot distinguish between single hyphens found in the running text and within parts of the input scanned in expand-only mode when searching for numbers in horizontal mode. Specifications like `\looseness=-1` won't be processed properly unless some special care is taken. One remedy might be to steal the commands which can suffer from this and to define them as commands which switch the hyphen-splitting feature off and reinstall it again using `\afterassignment`:

```
\let\@looseness=\looseness
\def\looseness{%
            \if@splithyphens
               \standardhyphens
               \afterassignment
                        \splithyphens
            \fi
            \@looseness}
```

Conclusion

We have outlined the way how the concept of font selection scheme used within LaTeX 2_ε allows to define alternative encoding schemes corresponding to a particular language and to generate formats comprising all of them. This is particularly useful in situations where multiple users ahare a common LaTeX installation but for various reasons use different code pages. One of the motivations for the present work was the situation with "official" encodings for Czech where CSTUG officially adopted ISO Latin 2 for Czech characters encoding, yet the LaTeX 2_ε standard is the Cork DC scheme. Besides of those two, other codings still persist for historical reasons. The principal declarations in the configuration files for this environment look as follows: Within `fontdef.cfg`, additional font encodings are defined and the `@hyphenation` hook is introduced. Note that ISO Latin2 is not `\lccode`-compatible with T1, and so changes of the code parameters must be done as well—that's what `\@@XXXCodes` commands are here for:

```
\DeclareFontEncoding{T1}%
          {\csname @@T1Codes\endcsname
           \@hyphenation{T1}}{}
\DeclareFontEncoding{OT1}{}{}
\DeclareFontEncoding{KOI}%
          {\@@KOICodes
           \@hyphenation{KOI}}{}
\DeclareFontEncoding{IL2}%
          {\csname @@IL2Codes\endcsname
           \@hyphenation{IL2}}{}
\DeclareFontEncoding{OML}{}{}
\DeclareFontEncoding{OMS}{}{}
\DeclareFontEncoding{OMX}{}{}
\DeclareFontEncoding{U}{}{\noaccents@}
```

The actual languages are introduced in configuration file `lthyphen.cfg` (declarations of the operations needed are skipped out) and the table for two versions of English without the possibility of using the split-hyphen feature, and two languages allowing splitting: Czech and Slovak, each of them for three different font encodings:

```
\DeclareLanguage{\english}%
               {hyphen.tex}%
               {\lefthyphenmin=2
                \righthyphenmin=3
                \@splitrequestedfalse}
\DeclareLanguage[T1,KOI,IL2]{\czech}%
               {czhyphen.tex}%
               {\lefthyphenmin=2
                \righthyphenmin=3
                \@requesthyphens}%
\DeclareLanguage[T1,KOI,IL2]{\slovak}%
               {skhyphen.tex}%
               {\lefthyphenmin=2
                \righthyphenmin=3
                \@requesthyphens}%
\DeclareLanguage{\british}%
               {ukhyphen.tex}%
               {\lefthyphenmin=2
                \righthyphenmin=3
                \@splitrequestedfalse}%
```

We have presented several requirements for such a multi-lingual multi-encoding system and showed the way how to implement them. The concept of environment definition when switching between languages is similar to that introduced within Babel, and indeed within Babel-based system it is possible to use the approach shown within this paper as a lower-level interface to language switching combined with encoding switching and the corresponding representation of hyphenation patterns. Configuration files with local modifications based on this approach can be found on `ftp.muni.cz` in `/pub/tex/cstug/csltx2e`.

Acknowledgment

This work has been supported by GA ČR grant No. 201/93/1269.

References

[1] J. Braams: Babel, a multilingual style-option system for use with LaTeX's standard document styles. TUGboat, Vol. 12 (1991), No. 2, 291–302.

[2] M. Goosens, F. Mittelbach, and A. Samarin: The LaTeX Companion. Addison-Wesley, Reading, MA, 1994.

[3] L. Lamport: LaTeX: a document preparation system, 2nd ed. Addison-Wesley, Reading, MA, 1994.

[4] LaTeX3 Project Team: LaTeX 2$_\varepsilon$ distribution. CTAN archives, June 1994.

◇ Jiří Zlatuška
Faculty of Informatics
Masaryk University
Burešova 20
602 00 Brno, Bohemia
zlatuska@informatics.muni.cz

```
\bgroup\catcode`-\active\expandafter\egroup\expandafter
\def\noexpand-{\ifx\protect\relax
                    \ifhmode
                        \ifinner
                            \expandafter\expandafter\expandafter\expandafter
                            \expandafter\expandafter\expandafter-
                        \else
                            \expandafter\expandafter\expandafter\expandafter
                            \expandafter\expandafter\expandafter\firstdash
                        \fi
                    \else%
                        \expandafter\expandafter\expandafter-
                    \fi
               \else
                    \expandafter\string\expandafter\string-
               \fi}
\def\onehyphen{-}
\def\splithyphen{\ifnum\expandafter\hyphenchar\the\font>-1
                       \discretionary{\char`-}{\char\hyphenchar\the\font}{\char`-}
                 \else
                       \discretionary{\char`-}{\char`-}{\char`-}
                 \fi}
\def\boxhyphen{\hbox{-}}
\def\twohyphens{--}
\def\threehyphens{---}
\catcode`\-=\active
\def\firstdash{\exfilerc
                   \seconddash
               \else
                    \ifnum\expandafter\hyphenchar\the\font=`-
                        \ifnum\righthyphenmin>2
                            \if l\nextchar
                                \boxhyphen      %   make sure we
                            \else               %   don't chop off `-li'
                                \splithyphen
                            \fi
                        \else
                            \splithyphen
                        \fi
                    \else
                        \onehyphen
                    \fi
               \fii}
\def\seconddash#1{\ifnexthyphen \thirddash  \else \twohyphens \fii}
\def\thirddash#1{\ifnexthyphen \manydash  \else \threehyphens \fii}
\def\manydash#1{\threehyphens\firstdash}
\def\ifnexthyphen#1\else#2\fii{%
            \def\comparechars{\ifx -\nextchar
                                  \def\next{#1}%
                              \else
                                  \def\next{#2}%
                              \fi
                              \next}%
           \futurelet\nextchar\comparechars}
\catcode`-=12
```

Figure 2: Hyphen-splitting code