Second Version of encTeX: UTF-8 Support

Petr Olšák
Czech Technical University in Prague
petr@olsak.net

Abstract

The UTF-8 encoding keeps the standard ASCII characters unchanged and encodes the accented letters of our alphabets in two bytes. The standard 8-bit TeX is not ready for the UTF-8 input because it has to manage the single character as two tokens. It means you cannot set the `\catcode`, `\uccode`, etc., of these single characters and you cannot do `\futurelet` of the next character in the normal way. The second version of my encTeX solves these problems.

The encTeX program is fully backward compatible with the original TeX. It adds ten new primitives by which you can set or read the conversion tables used by the input processor of TeX or used during output to the terminal, log and `\write` files.

The second version creates the possibility of converting the multi-byte sequences to one byte or to a control sequence. You can implement up to 256 UTF-8 codes as one byte and an unlimited number of other UTF-8 codes as a control sequence. All internals in 8-bit TeX work as usual if the normal "one byte encoding" of input files is used.

I think that the UTF-8 encoding will be used more commonly in the future. In such a situation, there is no other way than to modify the input processor of TeX; otherwise, the 8-bit TeX will be dead in a short time.

Résumé

Le codage UTF-8 garde les caractères ASCII inchangés et encode les lettres accentuées de nos alphabets en deux octets. Le TeX à 8 bits standard n'est pas prêt pour une entrée UTF-8 car il doit gérer les caractères comme deux octets. Cela signifie que vous ne pouvez pas changer `\catcode`, `\uccode`, etc. de ces caractères et vous ne pouvez pas faire un `\futurelet` du caractère qui suit, dans la sens habituel. La deuxième version de mon encTeX résoud ces problèmes.

encTeX est totalement compatible avec le TeX original. Il ajoute dix nouvelle primitives par lesquelles vous pouvez établir ou lire les tables de conversion utilisées par le processeur d'entrée de TeX ou utilisées pendant la sortie au terminal, et aux fichiers log et `\write`.

La seconde version donne la possibilité de convertir des séquences multi-octets vers un octet ou une commande. Vous pouvez implémenter jusqu'à 256 codes UTF-8 comme un octet and un nombre illimité de codes UTF-8 comme commandes. Toute la machinerie interne de TeX fonctionne comme si les fichiers d'entrée sont dans un «codage normal à un octet».

L'auteur pense que le codage UTF-8 va être de plus en plus courant dans l'avenir. Dans cette situation il n'y a pas d'autre moyen que de modifier le processeur d'entrée de TeX, sinon la version originale de TeX à 8 bits va disparaître sous peu.

What is encTeX?

EncTeX is a TeX extension which allows re-encoding of input stream in the input processor of TeX (before tokenization) and backward re-encoding of output stream during `\write` and output to the terminal and log. It is implemented as a patch to the change file `tex.ch`. The patches are ready for Web2C distribution on [1] and (maybe) encTeX will become a standard Web2C extension, as it is for MiKTeX. Try to use the `-enc` option on command line to test if your TeX is equipped with this extension. If not, you can get and apply the patches and rebuild TeX binaries. The patches affect TeX, eTeX, pdfTeX and pdfeTeX programs. All of these programs can make use of this extension.

The first version of encTeX was released in 1997. This version only implemented byte to byte conversion, by modifying TeX's internal *xord* and *xchr* vectors. EncTeX introduced three primitives in its first version: `\xordcode` (reads or sets the values of *xord* vector for input re-encoding), `\xchrcode` (reads or sets the values of *xchr* vector for output re-encoding) and `\xprncode` (reads or sets the values of newly introduced *xprn* vector which controls the "print-ability" of characters—it controls the possibility of the character conversion to `^^ab` form on output side). See my article [2] for more details.

The first version of encTeX was not widely used because TCX tables were re-introduced in the Web2C distribution immediately after encTeX was released. Roughly speaking, the TCX tables do the same job as the first version of encTeX, although less flexibly. There was no reason to combine the TCX tables with encTEX.

The second version of encTEX was designed and prepared by me in December 2002 and released in January 2003. This version introduces seven more primitives so users can control the multi-byte input re-encoding and reverse output re-encoding. Groups of bytes on input stream can be converted to one byte or to a control sequence. The conversion is done before tokenization, but a control sequence generated by this conversion is not re-tokenized again and the token processor does not go to the "ignoring spaces" state after such a control sequence. The backward conversion during `\write` allows you to convert one byte or a control sequence to the original group of bytes.

The second version of encTEX is backward compatible with the first one, of course. Detailed documentation is available [1]. Very nice on-line html documentation written by David Nečas (Yeti) is also available [5].

Motivation

I am the maintainer of a `csplain` format—the basic part of the CSTEX package (for Czech and Slovak users). `csplain` is similar to the very well-known plain TEX format (by Don Knuth, [4]). Moreover, `csplain` solves the processing of all letters from Czech and Slovak alphabets. It means that CS-fonts (encoded by ISO-8859-2) are used by default instead of the Computer Modern fonts, the hyphenation tables for Czech and Slovak languages are input in the same encoding and all Czech and Slovak letters are treated as single non-composite symbols. These symbols have their `\catcode`s set to 11 (letter), thus they can be used in control sequences too.

Czech and Slovak alphabets are encoded by many mutually incompatible standards and pseudo-standards in various operating systems and environments. All these encodings have to be converted internally to ISO-8859-2 in `csplain` at the input processor level and converted back to the input encoding during `\write`, terminal and log output. Only this rule preserves the independence of the TEX processing from the operating system.

If the source text of the Czech or Slovak document is transported from one environment to another, re-encoding to the standard of the target environment must be done, either automatically or manually by the user. The main principle is that the Czech and Slovak characters in source text have to be displayed correctly by the operating environment before the document is processed by `csplain`.

I created the `cstrip` test in 1998 [3]. You can verify if you are really using the `csplain` format with this test. It verifies if TEX's input processor is set correctly depending on your operating environment: all Czech and Slovak characters have to be mapped into ISO-8859-2 and they have to be written back to the input encoding on terminal, log and `\write` files. The `^^ab` form is not permitted for Czech and Slovak letters.

We were able to set the input processor properly for `csplain` in old TEX distributions. For example, emTEX used TCP tables. On the other hand, the Web2C distribution disabled its similar TCX tables in 1997, thus users were not able to implement the `csplain` format correctly in operating environments where different encoding of our alphabets from ISO-8859-2 were used. This was the main motivation of encTEX extension of TEX.

Now, the new encoding standard UTF-8, derived from Unicode, is used very often. The non-ASCII characters are encoded in two or more bytes. If this encoding standard is used in our operating environment then we need to be able to set multi-byte conversion in the input processor of TEX. There is no other way to carry out the `cstrip` test. This was my motivation for the second version of encTEX.

Multi-byte re-encoding

The detailed documentation is included in the encTEX package. Thus, only a short overview of the principles is presented here.

The second version of encTEX introduces seven new TEX primitives to define and control re-encoding between multi-byte input/output and TEX's internal representation. These are:

- `\mubyte` and `\endmubyte` primitives defining the conversions,
- `\mubytein`, an integer register controlling input conversion,
- `\mubyteout`, an integer register controlling output conversion,
- `\mubytelog`, an integer register controlling output to terminal and log file,
- `\specialout`, an integer register controlling `\special` argument treatment, and
- `\noconvert`, a primitive suppressing output conversion.

The default values of all the new registers are such that encTEX behaves compatibly with unmodified TEX (incidentally, it means zeroes).

You can set the conversion table via the pair of primitives `\mubyte` and `\endmubyte`. Examples:

```
\mubyte ^^c1   ^^c3^^81\endmubyte % Á
\mubyte ^^c4   ^^c3^^84\endmubyte % Ä
```

It means that, for example, the group of two bytes `^^c3^^81` will be converted to one byte `^^c1` (if `\mubytein` is positive) and this byte is converted back to byte sequence `^^c3^^81` during `\write` (if `\mubyteout` is positive) and to log and terminal (if `\mubytelog` is positive).

If your operating environment uses UTF-8 encoding then the two bytes `^^c3^^81` are displayed as Á. You can do the "normal things" with this character in your text editor:

```
\catcode 'Á=11  \def\myÁsequence{...}
...
\def\run{\futurelet \next \dotest}
\def\dotest{\ifx \next Á...}
\run Áha
...
\uccode'Á='Á \lccode'Á='á \sfcode'Á=999
...
```

This behavior is very desirable for the `csplain` format and `cstrip` test. You can convert your old `csplain` documents to the new UTF-8 encoding and you can process them with `csplain` in operating environments supporting UTF-8. You get absolutely the same result as in the old days. This backward compatibility is most important for me.

Next example:

```
\mubyte \Alpha      ^^ce^^91\endmubyte
\mubyte \Beta       ^^ce^^92\endmubyte
...
\mubyte \leftarrow ^^e2^^86^^90\endmubyte
\mubyte \uparrow   ^^e2^^86^^91\endmubyte
...
```

For instance, the group of three bytes `^^e2^^86 ^^90` is now converted to the `\leftarrow` control sequence and this control sequence is converted back to `^^e2^^86^^90` during `\write` if `\mubyteout` ≥ 3. The UTF-8 encoding of math characters is implemented in this way; see the `utf8raw.tex` file in the encTEX distribution, and `math-example.tex` for more complex examples.

The UTF-8 encoding tables for encTEX were prepared by David Nečas [6]. He has made his own Python script which converts the `NamesList.txt` [7] with Unicode declarations of characters to the `\mubyte` ... `\endmubyte` tables. This script is included in the encTEX distribution.

There is another way to declare math symbols:

```
\mubyte \utfAlpha   ^^ce^^91\endmubyte
...
\def\utfAlpha{\ensuremathmode \Alpha}
...
\def\ensuremathmode #1{\ifmmode #1\else
   $#1$\fi}
```

This second solution is more robust because you can write math symbols in the UTF-8 encoding without needing to start math mode explicitly. Note that these symbols are displayed as natural math symbols in your text editor. I did not use this solution in my macros distributed with encTEX because this concept is not compatible with common TEX documents where all math mode switches are explicitly written.

More funny examples

You can use encTEX capabilities for purposes other than character encodings. Consider this next simple example:

```
\mubyte \TeX         TeX\endmubyte
\mubyte \copyright  (C)\endmubyte
\mubyte \dots        ...\endmubyte
```

If you write "`TeX and friends`" (without a backslash) then input processor of encTEX converts this stream to `\TeX`, ⟨*space*⟩, `a`, `n`, `d`, ⟨*space*⟩, `f`, etc. This is the desired behavior. Moreover, if `\mubyteout` ≥ 3 then the `\TeX` control sequence is not expanded during `\write`, but rather is converted back to its input byte sequence "`TeX`". On the other hand, if you write `\LaTeX`, then the input is converted to two control sequences `\La\TeX`, which is not desired. You can solve this problem by defining the "`\La`" macro or declaring:

```
\mubyte \LaTeX       LaTeX\endmubyte
\mubyte \LaTeXe      LaTeX2e\endmubyte
```

Note that both byte sequences in this example begin by the same text "`LaTeX`". If the two characters "`2e`" follow immediately then a `\LaTeXe` control sequence is generated (by the second line of this example) else the `\LaTeX` control sequence is generated. (The order of the lines in this example is unimportant.)

What happens, if this setting is active and you write `\LaTeX` (including the backslash)? Nothing bad. The empty control sequence before the generated control sequence `\LaTeX` is suppressed by encTEX, thus only the `\LaTeX` control sequence is output.

I implemented a program `vlna` for adding tildes after the Czech one-letter prepositions (v, k, s, u, o, z) entirely in encTEX using `\mubyte`. It correctly handles math mode (no tildes are added there). It's available in the encTEX distribution as an example of crazy applications of encTEX in the file `vlna.tex`.

References

[1] `http://www.olsak.net/enctex.html`, the main page of the encTEX project.

[2] Petr Olšák: *EncTEX—A little extension of TEX*, in: *TUGboat* 19(4), (1998), pp. 336–371.

[3] `ftp://math.feld.cvut.cz/pub/cstex/base/cstrip.tar.gz`.

[4] Donald Knuth: *The TEXbook*.

[5] `http://trific.ath.cx/tex-mf/enctex/`

[6] `http://trific.ath.cx/`, David Nečas's home page.

[7] `http://www.unicode.org/Public/UNIDATA/NamesList.txt`