

UTF-8 installations of T_EX

Igor Liferenko

Abstract

In its design T_EX has the concepts of “internal encoding” and “external encoding”. This fact allows T_EX to work with any encoding.

We use Unicode as T_EX’s external encoding. Then we change the necessary parts of T_EX to use UTF-8 as the input/output encoding.

The resulting implementation passes the `trip` test.

1. Initialization

Note: we use the `web2w` [1] implementation of T_EX, but the ideas described here can be applied to any implementation.

First, we change the data type of characters in text files to `wchar_t` to accommodate Unicode values.

Background: this predefined C type allocates four bytes per character (on most systems). Character constants of this type are written as `L'...'` and string constants as `L"..."`.

(For brevity, in the diffs following, the original code from `web2w`’s `tex.w` source is preceded with `<` characters, and the new code with `>`. Both are sometimes reformatted for presentation in this article, and for readability we sometimes leave a blank line between the pieces. The actual implementation uses the file `utex.patch` [2].)

```
< @d text_char unsigned char
> @d text_char wchar_t
```

Use values from the basic multilingual plane (BMP) of Unicode.

```
< @d last_text_char 255
> @d last_text_char 65535
```

Then we change the size of the `xord` array [3] to 2^{16} bytes.

```
< ASCII_code xord[256];
> ASCII_code xord[65536];
```

Elements in the `xchr` array [3] are overridden using the file `mapping.w`.

```
@i mapping.w
```

This file specifies the character(s) required for a particular installation of T_EX, for example:

```
xchr[0xf1] = L'ë';
```

A complete example of `mapping.w` is here:

<https://github.com/igor-liferenko/cweb>

`TEX_format_default` is in T_EX’s external encoding.

```
< ASCII_code TEX_format_default
< [1+format_default_length+1]
< =" TeXformats/plain.fmt";

> wchar_t TEX_format_default
> [1+format_default_length+1]
> =L" TeXformats/plain.fmt";
```

It remains to specify that the C library functions for conversion to and from Unicode work with UTF-8:

```
setlocale(LC_CTYPE, "C.UTF-8");
```

and to add the necessary headers.

```
#include <wchar.h>
#include <locale.h>
```

2. Input

For automatic conversion from UTF-8 to Unicode, text files (including the terminal) must be read with the C library function `fgetwc` [4].

In the original `tex.w`, the macro `get` is used for reading text files, as well as `tfm` and `format` files.

Text files are read in the functions `a_open.in` and `input.ln`. In `a_open.in` we replace the macro `reset` with its expansion and then in both functions we change `get((f))` to `(f).d=fgetwc((f).f)`

3. Output

For automatic conversion from Unicode to UTF-8, text files (including the terminal) must be written with the C library function `fwprintf` [4].

In the original `tex.w`, in all cases but one, the macro `write` is used for writing text files. So, we change `fprintf` to `fwprintf` in the definition of `write`. The one case where `write` is used is for writing `dvi` files — there we just use its old expansion.

In addition to redefining the macro `write`, we need to add the ‘L’ prefix to strings which are used in the macros that call the macro `write`. These changes are trivial and there are quite a few of them so we will not list them here. Instead, we show the following cases, where the conversion specifier in the `printf`-style directives also needs to change:

```
< wterm("%c",xchr[s]);
> wterm(L"%lc",xchr[s]);

< wlog("%c",xchr[s]);
> wlog(L"%lc",xchr[s]);
```

```
< write(write_file[selector],"%c",xchr[s]);
> write(write_file[selector],L"%lc",xchr[s]);
```

4. The file name buffer

The name of the file to be opened, which is stored in the *name_of_file* buffer, must be encoded in UTF-8. Therefore, each character passed to *append_to_name*, before being added to *name_of_file*, must be converted to UTF-8. This is done using the C library function *wctomb* [4].

In the *append_to_name* macro, the variable *k* is used as the index into the *name_of_file* buffer where the last byte was stored. Originally, *k* was always increased and provisions were made that characters will not be written beyond the end of buffer (which has the index *file_name_size*); *name_length* was then set to the minimal value between *k* and *file_name_size*.

We cannot do the same in our implementation, because we may reach the end of the buffer in the midst of a multibyte character. Instead, if the next multibyte character does not fit into the buffer, we prevent *k* from being increased by negating its value:

```
< @d append_to_name(X) { c=X;incr(k);
<   if (k <= file_name_size)
<     name_of_file[k]=xchr[c]; }

> @d append_to_name(X) { c=X;
>   if (k >= 0) { /* try to append? */
>     char mb[MB_CUR_MAX];
>     int len = wctomb(mb, xchr[c]);
>     if (k+len <= file_name_size)
>       for (int i = 0; i < len; i++)
>         name_of_file[i+k] = mb[i];
>     else
>       k = -k; /* freeze k */ } }
```

In *pack_file_name* and *pack_buffered_name* (the functions that call *append_to_name*), we have to “un-freeze” its value if it was “frozen”.

```
if (k < 0) k = -k;
```

In *make_name_string*, each (multibyte) character from *name_of_file* must be converted from UTF-8 to Unicode, before being converted to T_EX’s internal encoding. This is done using the C library function *mbtowl* [4].

```
< append_char(xord[name_of_file[k]]);

> { wchar_t wc;
>   k += mbtowl(&wc, name_of_file+k,
>             MB_CUR_MAX) - 1;
>   append_char(xord[wc]); }
```

In the code checking *format_default_length* for consistency, we use the C library function *wcstombs* [4] to count the number of bytes in the UTF-8 representation of *TEX_format_default*.

```
< if (format_default_length > file_name_size)

> if (wcstombs(NULL,TEX_format_default+1,0) >
>     file_name_size)
```

In the function *pack_buffered_name*, the code that drops excess characters assumes that each character is one byte. But the number of bytes used to represent a character in UTF-8 may be more than one. Therefore, after appending the contents of *buffer[a .. b]* to *name_of_file*, we roll back in it character by character until the format extension fits in it. We use the C library function *mblen* [4] to determine the start of a multibyte character to be discarded.

```
while (k+wcstombs(NULL,TEX_format_default+
format_default_length-format_ext_length
+1,0) > file_name_size) { k--;
while (mblen(name_of_file+k+1,MB_CUR_MAX)==
-1) k--; }
```

This code in *pack_buffered_name* becomes unnecessary and is removed:

```
if (n+b-a+1+format_ext_length >
file_name_size)
b=a+file_name_size-n-1-format_ext_length;
```

References

- [1] Ruckert, Martin. WEB to cweb.
hint.userweb.mwn.de/hint/web2w.html
- [2] Source of the present implementation.
<https://github.com/igor-liferenko/tex>
- [3] Knuth, Donald E. *T_EX: The Program*, 1986. ISBN 0201134373.
- [4] Single Unix Specification. Introduction to ISO C Amendment 1 (Multibyte Support Environment).
http://unix.org/version2/whatsnew/login_mse.html

◇ Igor Liferenko
igor.liferenko (at) gmail dot com