The newunicodechar package*

Enrico Gregorio
Enrico dot Gregorio at univr dot it

November 12, 2012

1 Introduction

When using Unicode input with LaTeX it's not so uncommon to get an incomprehensible error message such as

```
Unicode char \u8:xxx not set up for use with LaTeX
```

where `xxx` may be the actual character we input or a combination of strange characters. This happens because the `utf8` option given to inputenc or inputenx defines the LaTeX meaning of many Unicode characters, but, of course, not all of them.

For example, one might want to write some Latin words with prosodic marks, i.e., the diacritics that tell whether a vowel is long or short, in order to distinguish between 'pŏpŭlus' (people) and 'pōpŭlus' (poplar), but using the actual Unicode characters that make the LaTeX document easier to read; look at the following table, where on the left the input is via the LaTeX Internal Character Representation (LICR) and on the right it's via Unicode characters,

LICR	Unicode
`p\u{o}p\u{u}lus`	pŏpŭlus
`p\={o}p\u{u}lus`	pōpŭlus

and judge by yourselves which one is better. Unfortunately, the `utf8` option to inputenc doesn't define a meaning for `ŏ`, `ō`, and `ŭ`. As a matter of fact, only `ă` and `Ă` are defined, as they are used in the Romanian language.

One might resort to `\DeclareUnicodeCharacter` in the document's preamble, but this requires looking up at the (long) list of Unicode characters and jotting down the relevant numbers. For example, `ŏ` is `U+014F`, so the declaration

```
\DeclareUnicodeCharacter{014F}{\u{o}}
```

*This document corresponds to newunicodechar v1.1, dated 2012/11/12.

would do for ŏ.

The present package introduces a simpler interface that frees the user from the burden to look up in the tables: all it's needed is

```
\newunicodechar{ŏ}{\u{o}}
```

You are not restricted to definition like this: for example, ♪ is Unicode U+266A, but you are not required to know it: if your editor can insert the character ♪, you may define its meaning by loading a package that provides it and say

```
\usepackage{wasysym}
\newunicodechar{♪}{\eighthnote}
```

A similar problem may arise even with XeLaTeX. A frequently asked question on mailing lists or discussion groups is how to print some particular character in a different font than the main one of the document, say, for example, the Euro sign which, in some fonts, is horrible. The usual answer is to write something like

```
\newfontfamily{\eurofont}{⟨some font⟩}
\catcode`€=\active
\protected\def €{{\eurofont\char`\€}}
```

which, for the average user, is somewhat scaring. With newunicodechar this may be simplified into

```
\newfontfamily{\eurofont}{⟨some font⟩}
\newunicodechar{€}{{\eurofont\texteuro}}
```

2 Usage

The package requires the use of a Unicode engine, i.e., XeLaTeX or LuaLaTeX, or, with (pdf)LaTeX, the inputenc or inputenx package along with the utf8 option. It won't work with the utf8x option that employs a completely different mechanism for parsing Unicode characters in (pdf)LaTeX. It should be said that utf8x defines many more characters than utf8, so that the present package wouldn't be needed.

Of course the LaTeX document must be written using a Unicode savvy editor.

verbose

The package has only one option, verbose, which is off by default. If the package is called by \usepackage[verbose]{newunicodechar}, then the informative message on the log file will show the old definition along with the warning about the redefinition. Unfortunately this definition usually has a rather cryptic format; for example, redefining ă would print

```
Redefining Unicode character; it meant
***  \IeC {\u a}  ***
before your redefinition on input line 22.
```

Call the package with this option if you are worried about what you are redefining, but the meaning of the Unicode character should correspond easily to just one LICR entry. This option does nothing when a Unicode engine is used, because no character is active initially (except for ~), so there should be non risk to redefine anything.

`\newunicodechar` The package provides only one command, `\newunicodechar`, which must be called with two arguments:

`\newunicodechar{`⟨*char*⟩`}{`⟨*code*⟩`}`

where ⟨*char*⟩ is the Unicode character to which we need to give a meaning and ⟨*code*⟩ is that meaning, that is the LaTeX code that will be substituted to the character. Here is what's needed for the prosodic marks in Latin:

```
%\newunicodechar{Ă}{\u{A}} \newunicodechar{ă}{\u{a}}
\newunicodechar{Ĕ}{\u{E}} \newunicodechar{ĕ}{\u{e}}
\newunicodechar{Ĭ}{\u{I}} \newunicodechar{ĭ}{\u{\i}}
\newunicodechar{Ŏ}{\u{O}} \newunicodechar{ŏ}{\u{o}}
\newunicodechar{Ŭ}{\u{U}} \newunicodechar{ŭ}{\u{u}}
\newunicodechar{Ā}{\={A}} \newunicodechar{ā}{\={a}}
\newunicodechar{Ē}{\={E}} \newunicodechar{ē}{\={e}}
\newunicodechar{Ī}{\={I}} \newunicodechar{ī}{\={\i}}
\newunicodechar{Ō}{\={O}} \newunicodechar{ō}{\={o}}
\newunicodechar{Ū}{\={U}} \newunicodechar{ū}{\={u}}
```

The first line is commented out, because those characters are already defined. It doesn't hurt to give again a definition, LaTeX will just warn about it.

Caution: when used with XeLaTeX or LuaLaTeX, there will be no warning, but the standard setup doesn't define any active character.

The first argument *must* consist of a single Unicode character, but the package checks for it and raises an error otherwise; it will raise an error also if the first argument consists of a plain ASCII character: defining them is not allowed in (pdf)LaTeX and would break almost everything in XeLaTeX or LuaLaTeX.

3 An easier way?

One could dispense with this package, since the same effect may be obtained in the way we have already seen in XeLaTeX or LuaLaTeX; with (pdf)LaTeX, calling `\newunicodechar{ū}{\={u}}` is equivalent to say

```
\makeatletter
\@namedef{u8:\detokenize{ū}}{\={u}}
\makeatother
```

4 Implementation

The usual presentation, that we repeat here for completeness.

```
\ProvidesFile{newunicodechar.dtx}
\NeedsTeXFormat{LaTeX2e}[2008/04/05]
\ProvidesPackage{newunicodechar}
```

The date for the format has been chosen to match the last version of the `utf8enc.dfu` file that provides definitions for Unicode characters.

Now the real macros. First of all we check that the typesetting engine is sufficiently recent to include ε-TEX extensions.

```
\@ifundefined{eTeXversion}
  {\PackageError{newunicodechar}{LaTeX engine too old, aborting}
   {Please upgrade your TeX system}\@@end}{}
```

4.1 Options

There's only one option.

```
\DeclareOption{verbose}{\let\nuc@verbose=T}
\ProcessOptions\relax
```

4.2 Error messages

```
\def\nuc@onebyteerr{\PackageError{newunicodechar}
  {ASCII character requested}
  {Only characters above U+007F may be defined; you asked
   for\MessageBreak a plain ASCII character and your definition
   has been ignored.}}
\def\nuc@emptyargerr{\PackageError{newunicodechar}
  {Empty argument}
  {You shouldn't write \protect\newunicodechar{}{...}}}
\def\nuc@invalidargerr{\PackageError{newunicodechar}
  {Invalid argument}
  {The first argument to \protect\newunicodechar\space is
   either\MessageBreak too long or an invalid sequence of bytes}}
```

4.3 Unicode engines

In case we are running a Unicode engine (`xelatex` or `lualatex`), the definition of the main macro is easier: we check whether the character is above 127 and, in this case, we make it active expanding to the second argument. This definition of the main macro will be seen only if the engine has the `^^^^` convention for inputting characters in hexadecimal format, so that `^^^^0021` is one token and `\@gobble` will eat it up, making `\next` equal to `\@empty`; with an eight bit engine (`latex` or `pdflatex`), `\@gobble` will swallow only `^^^`.

In this case we define the main macro all in one swoop; first we check that the first argument is nonempty, then we act only if it consists of only one (character) token. Then if the charcode of this token is less than 127 we emit an error message; otherwise we activate the character and define its (protected) expansion to be the second argument. The last action, in this case, is to allow `\newunicodechar` only in the preamble. Then we do `\endinput`. If the engine is not Unicode savvy, everything up to the closing `\fi` is swallowed up.

\newunicodechar Here is the code for defining the Unicode engine version of the main macro. Notice that we try being on the safe side by not assuming any particular category code for ~, because we restore it after giving the definition where we need it to be active.

```
\begingroup
\catcode`\^=7 \catcode30=12 \catcode`\!=12 % for safety
\edef\next{\@gobble^^^^0021}
\expandafter\endgroup
\ifx\next\@empty % Start of code for Unicode engines
\chardef\nuc@atcode=\catcode`\~
\catcode`\~=\active
\def\newunicodechar#1#2{%
  \if\relax\detokenize{#1}\relax
    \nuc@emptyargerr
  \else
    \if\relax\detokenize\expandafter{\@cdr#1\@nil}\relax
      \ifnum`#1>\string"7F
        \catcode`#1=\active
        \begingroup\lccode`\~=`#1
        \lowercase{\endgroup\protected\def~}{#2}%
      \else
        \nuc@onebyteerr
      \fi
    \else
      \nuc@invalidargerr
    \fi
  \fi}
\catcode`\~=\nuc@atcode
\@onlypreamble\newunicodechar
\expandafter\endinput
\fi % End of code for Unicode engines
```

4.4 Eight bit engines

From now on we can assume an eight bit engine is used; we check that inputenc or inputenx has been loaded with the right option, otherwise we define \newunicodechar to swallow its arguments, after a warning. We need to first check for inputenx because it loads inputenc.

```
\def\nuc@stop{\PackageWarningNoLine{newunicodechar}
  {This package won't work without loading\MessageBreak
   `inputenc' or `inputenx' with the `utf8' option}%
  \let\newunicodechar\@gobbletwo\endinput}
\@ifpackageloaded{inputenx}
  {\def\nuc@tempa{inputenx}}
  {\@ifpackageloaded{inputenc}{\def\nuc@tempa{inputenc}}{\nuc@stop}}
\@ifpackagewith{\nuc@tempa}{utf8}{}{\nuc@stop}
\@ifpackagewith{\nuc@tempa}{utf8x}{\nuc@stop}{}
```

\newunicodechar The main macro. We set the temporary switch to false and put in \nuc@tempa the first argument, but detokenized, since it consists of active characters. We

check that it's not empty and access to its first token that we put into `\@tempb`. Then we call `\nuc@check` and if it sets the temporary switch to true, we execute the definition, since the first argument is a valid UTF-8 character, but we issue a warning if it already has a meaning. All we need to do is to define the macro `\csname u8:\nuc@tempa\endcsname`, because that's how inputenc acts. The part between `\ifdefined` and the corresponding `\fi` will be executed only with the `verbose` option.

```
\def\newunicodechar#1#2{%
  \@tempswafalse
  \edef\nuc@tempa{\detokenize{#1}}%
  \if\relax\nuc@tempa\relax
    \nuc@emptyargerr
  \else
    \edef\@tempb{\expandafter\@car\nuc@tempa\@nil}%
    \nuc@check
    \if@tempswa
      \@ifundefined{u8:\nuc@tempa}{}
        {\PackageWarning{newunicodechar}
          {Redefining Unicode character\ifdefined\nuc@verbose;
           it meant\MessageBreak
           ***\space\space\nuc@meaning\space\space***\MessageBreak
           before your redefinition\fi}}%
      \@namedef{u8:\nuc@tempa}{#2}%
    \fi
  \fi
}
```

The first helper macro computes the number of bytes in the first argument, though it appears to the user as a single character. The third is used for the `verbose` option.

```
\def\nuc@getlength#1{%
  \ifx#1\@nil
    \expandafter\relax
  \else
    +1\expandafter\nuc@getlength
  \fi}
\ifdefined\nuc@verbose
  \def\nuc@meaning{\expandafter\expandafter\expandafter
    \strip@prefix\expandafter\meaning\csname u8:\nuc@tempa\endcsname}
\fi
```

We select the action based on the length of the first argument; in case only one byte appears, the user is trying to define an ASCII character, which is not allowed; if the input is two, three, or four bytes long, we check whether the first byte has the correct form: its binary form must be, respectively, `110xxxxx`, `1110xxxx`, or `11110xxx`, so not less than 192, 224, or 240. The macro `\nuc@ch@ck` does precisely this, issuing an error message if the argument is invalid or else setting the temporary switch to true.

```
\def\nuc@check{%
```

```
  \ifcase\numexpr0\expandafter\nuc@getlength\nuc@tempa\@nil
    \or %0
    \nuc@onebyteerr\or %1
    \nuc@ch@ck{192}\or %2
    \nuc@ch@ck{224}\or %3
    \nuc@ch@ck{240}\else %4
    \nuc@invalidargerr
  \fi}
\def\nuc@ch@ck#1{%
  \expandafter\ifnum\expandafter`\@tempb<#1\relax
    \nuc@invalidargerr
  \else
    \@tempswatrue
  \fi
}
```

Finally we disallow `\newunicodechar` outside the preamble.

```
\@onlypreamble\newunicodechar
```

Change History

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.