The PracTEX Journal, 2012, No. 2
Article revision 2012/10/15

Some misunderstood or unknown LaTeX2ε tricks (VI)

Luca Merciadri

Email Luca.Merciadri@student.ulg.ac.be
Website http://www.student.montefiore.ulg.ac.be/~merciadri/

Abstract Customizing a document class is important in the LaTeX world. We will here see two examples of this through the `lettre` document class. Next, we will consider a small fraction of a bigger LaTeX problem: the encoding facts.

1 Introduction

Customizing a document class is important in the LaTeX world. We will here see two examples of this through the `lettre` document class:

2. Removing line bending,
3. Making the space smaller, between address, etc., and the beginning of the text.

Next, we will consider a small fraction of a bigger LaTeX problem: the encoding facts.

2 Customization of a document class: example with `lettre`

LaTeX document classes can be customized: macros can be (re)defined and lengths can be set, for example. This allows you to adapt an existing document class to your own use. That might come in handy when you have a document class that suits most of your document's needs, but for which you would like to change some specific aspects.

We will here take the example of the `lettre` document class to illustrate how these two things can be done.

2.1 Removing line bending

The `lettre` documentclass draws a small line bending to help you bending the letter in three parts once you have printed it. If you find this line bending disgracious, here are two methods to remove it [8]:

1. *Defining a macro*. Put

   ```
   \makeatletter
   \newcommand*{\NoRule}{\renewcommand*{\rule@length}{0}}
   \makeatother
   ```

   in the preamble. You can then use `\NoRule` after the beginning of the `letter` environment;

2. *Creating a new class*. You can also define a new class, `xletter.cls`, for example, defined like this:

   ```
   \NeedsTeXFormat{LaTeX2e}
   \ProvidesClass{xlettre}

   \newcommand*{\xlettre@do}{}
   \newcommand*{\xlettre@rule}{}
   \newcommand*{\xlettre@norule}{%
     \let \xlettre@institut=\institut
     \def \institut ##1{%
       \xlettre@institut{##1}%
       \def \rule@length {0}%
     }%
     \def \@institut {%
       \makeatletter \input{default.ins}\makeatother
       \def \rule@length {0}%
     }%
   ```

```
}

\DeclareOption{rule}{\let \xlettre@do =\xlettre@rule}
\DeclareOption{norule}{\let \xlettre@do =\xlettre@norule}
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{lettre}}

\ExecuteOptions{norule}
\ProcessOptions

\LoadClass{lettre}

\xlettre@do

\endinput
```

You can then use either

```
\documentclass[norule]{xlettre}
```

or

```
\documentclass{xlettre}
```

or

```
\documentclass[rule]{xlettre}
```

2.2 Making the space smaller, between address, etc., and the beginning of the text

If you use the `lettre` class, you might dislike the space between address and the beginning of the text of the letter. To make this space a lot smaller, that is, to *set the length* of `openingspace`, simply use

```
\setlength\openingspace{-1cm}
```

in the preamble. Thanks to Gonzalo Medina Arellano for this [5].

3 Encoding problems

If you are collaborating with other persons using different OSes, or simply migrating from one platform to another, you may have troubles with accents and special characters, especially if the language is French, Polish, or another language which makes an extensive use of special characters.

Some persons sometimes mix up the words which are related to *encoding*. We shall give here a brief summary of how you need to deal with encoding and LaTeX. Thanks for both Robin Fairbairns', Philipp Lehman's, Günter Milde's, Philipp Stephani's and Dominik Waßenhoven's contributions from which I inspired a lot at [2].

3.1 The `inputenc` package: the encoding of the document

You might have heard that putting

```
\usepackage[encoding]{inputenc}
```

in the preamble of every document you write should avoid encoding problems, assuming `encoding` is the encoding of the related `.tex` file. This is partially true, as it will solve *most* encoding problems, but not all.

3.1.1 Description

According to [7], the `inputenc` package maps certain characters to their corresponding TeX macros according to the encoding option you select.

3.1.2 Encoding choice

Consider the `encoding` parameter. If you choose to use `utf8x`, it will use the e*x*tended UTF-8 character set, and you will be able to type, among other languages, Greek. Although there are popularity differences, text encoding is nowadays unrelated to the operating system since all modern operating system components (windowing systems, font systems, drawing engines, ...) are Unicode-based or at least Unicode-capable. [2] For the rest of this text, remember that UTF-8 is an encoding for Unicode.

The utf8 encoding covers the code ranges for which there is a defined LaTeX encoding, while utf8x covers code ranges for which there is a usable font. [2] You might then think that utf8x is the best solution, but it has many disadvantages.

Amongst them, a first problem is that the UTF-8 decoder of utf8x is more intrusive than the one of utf8. utf8 has an expandable scanner, when the one in utf8x is non-expandable. There is more potential for conflicts with other packages in the latter case. [2]

A second problem is that utf8x uses the ucs package, which is no longer maintained, and which breaks important packages such as csquotes. [2]

3.1.3 Solutions?

You are definitely not lost, because there are also inputenx (a drop-in replacement of inputenc) and luainputenc (for 8-bit encoding on LuaTeX). For comparison, XeTeX and LuaTeX are natively in UTF-8 mode (and never require inputenc). [2] If you use pdfTeX and want basic UTF-8 support (e.g., for accented Latin characters), load utf8enc.def.

But, generally, utf8 lets you, for example, type Arabic script, Cyrillic script, Czech, French, German, (not Greek, use utf8x for example), Italian, Polish, Spanish, etc.

3.1.4 Platforms

On Microsoft Windows ©, users tend to use either ISO-8859-1 (which is commonly referred to as Latin-1), or CP1252. The former is generally intended for "Western European" languages. In this case, you need to replace encoding by latin1. The latter is an eight-bit character encoding designed to cover languages that use the Cyrillic alphabet such as Russian, Bulgarian, Serbian Cyrillic and other languages (French, ...), which do not use the Cyrillic alphabet. It is the most widely used for encoding the Bulgarian, Serbian and Macedonian languages, for example.

The character set CP1252 uses some of those positions for printable characters. Thus, the CP1252 character set is not identical with ISO-8859-1, because CP1252 contains more symbols than ISO-8859-1. [3] This is why using latin1 as an option of inputenc under Microsoft Windows © pose no problem.

If you want to stick with `inputenc`, you might also use `ansinew` for `inputenc`. Sticking with `latin1` should not pose any problem until you type in alphabets that `latin1` does not understand, such as Cyrillic. In ISO-8859-1, code positions $128 - 159$ are explicitly reserved for control purposes; they "correspond to bit combinations that do not represent graphic characters."

That being said, as Unicode is state of the art, it is disadviced to continue using either CP1252 or ISO-8859-1. It is suggested to use Unicode whenever possible. Moreover, since UTF-8 is supported in a better way by the `inputenc` package (and thus by pdfTeX-based LaTeX documents) than other encodings, it is the only choice that can be recommended. As a preliminary conclusion, you would then better use `utf8` as `encoding` value for `inputenc`.

If you are switching to `utf8` because you are sharing LaTeX sources with others and want to avoid problems with `Latin1/15` vs. `CP1252` vs. `MacRoman` (or similar for Eastern European encodings), using `inputenc` with `utf8` will work fine. This is, once again, another reason to use `utf8`.

If you need "real" Unicode support (e.g., when mixing scripts), you would better use a Unicode-savvy engine. More specifically, XeTeX and LuaTeX are the best options if you want UTF-8. As a conclusion, if you stick with normal LaTeX use, simply invoke `utf8` unless you really need `utf8x`, whatever your platform. [2, 3]

3.2 The proper encoding of the document file

To avoid clashes, the best thing is to keep your document in the same encoding as the encoding `encoding`, which is the option of `inputenc`. This is what we mentioned in the previous section. But, now, we will investigate the different ways to type texts in LaTeX.

3.2.1 Directly writing characters without commands

Directly writing characters without their associated commands (for example "é" written with a `e` and an acute accent) poses no problem until there is no encoding clash. If `encoding` for `inputenc` matches the document encoding, and that the current architecture understands this encoding, there will thus be no problem. [2]

But if you need to share sources without modifying the encoding, using commands at the place of direct keystrokes is better. But keep in mind that it makes the code more unreadable, and that this is also very unnatural to type such characters like this. [2]

Consider for example a text written under Microsoft Windows ©. Reading it on a Linux UTF-8 workstation, and saving it as UTF-8, will result in many encoding clashes, with exotic symbols.

There are actually four kinds of persons:

1. *Those who stick with commands.* Commands will always be valid, and, if deprecated one day, using `renewcommand` or other structures will make no problem. Less readable, but more portable, [2]

2. *Those who use commands only when necessary.* These are persons who try to see which character from their keyboard is directly rendered, which one is not, and, for those which are rendered, they typeset them directly, and, for those which are not rendered (as now), they use commands. This is sometimes tedious, difficult, error-prone, but it appears natural to assume that a character that is keyed in generates appropriate output unless it belongs to a special category (like \ or %): documents are easy to typeset, read, and edit for users with a similar keyboard, [2]

3. *Those who use various tricks to make LaTeX behaves as they want, even if they want things that are contrary to the state-of-the-art.* This might not be the best solution, but it depends on what the tricks are, [2]

4. *Those who use Unicode whenever possible.* While sometimes difficult to input (a good text editor will help), this results in readable sources that will work across OS boundaries. [2]

The best thing which can be recommended is evidently to stick with commands, such as demonstrated before. There are lots of sources to find symbols. A well-known one is [6]. One sometimes needs to include packages for other symbols, though.

3.2.2 Converting a file to the good encoding

If, say, you are dealing with documents in another encoding, the best thing is to use the following procedure, assuming you are working with Linux (or with such a virtual machine):

1. Find their current encoding,
2. Know what their future encoding will be (generally, you would better choose UTF-8 for aforementioned reasons),
3. Be sure that the encodings are compatible (i.e. the target character set is a superset of the source character set). If this is not the case, you will loose information,
4. Execute, a sample file being `fileinoldencoding.tex`, and the same file, in its new encoding being `fileinnewencoding.tex`:

   ```
   iconv -f oldencoding
   fileinoldencoding.tex -o
   fileinnewencoding.tex
   ```

   where `oldencoding` could be, for example, `windows-1252`. You might make this process automatic, *e.g.* by creating a shell file (here it is `bash`):

   ```
   #/bin/bash
   for i in *.tex
   iconv -f windows-1252 -t utf-8 -- "$i"
   > "$i.utf8" && mv -- "$i.utf8" "$i"
   done
   ```

   and executing this file in a folder containing `.tex` files. You can evidently modify this script or the aforementioned commands as you want, to use another encoding. By default, the encoding is `utf8` if this is your current locale, but if you want encoding `newencoding`, use

```
-t newencoding
```

5. Open the file in an editor, the editor being set to open files in the output encoding,

6. If you see strange characters, there is a problem, and check the procedure. If everything seems normal, you can modify the file, save the modifications, but everything under the new encoding,

7. Compile the file(s) with the good `inputenc` declaration, as explained above.

Note that other tools perform such operations, such as `recode`. [2]

3.3 Dealing with BiB files

Everything from this section comes from Philipp Lehman, Philipp Stephani, either from [2] or from [4].

If you are using BiBTEX, you might also have problems. The first problem is that BiBTEX *cannot* handle UTF-8 and non-fixed-width encodings in general. This is probably one of the most common misunderstanding when it comes to BibTEX. There is no way around this restriction. BibTEX cannot deal with multi-byte encodings.

You might have already heard about `bibtex8`, a drop-in BiBTEX replacement which supports 8-bit input. While it cannot handle UTF-8 either, it can sort 8-bit input in a way that is actually useful in languages other than English, when supplied with a suitable `csf` file.

Once again, you will need to use ASCII notation. Traditional BiBTEX can only alphabetize ASCII characters correctly. If the bibliographic data includes non-ASCII characters, they have to be given in ASCII notation. For example, instead of typing a letter like 'ä' directly, you need to input it as `\"a`, using an accent command and the ASCII letter. This ASCII notation needs to be wrapped in a pair of curly braces. Traditional BibTEX will then ignore the accent and use the ASCII letter for sorting.

Apart from it being inconvenient, there are two major issues with this convention. One subtle problem is that the extra set of braces suppresses the kerning on both sides of all non-ASCII letters. But, also, simply ignoring all accents may not

be the correct way to handle them, for alphabetical reasons which depend on the language.

These are the major reasons why switching to `bibtex8`, the 8-bit implementation of BibTEX, is advisable. It can sort in a case-sensitive way and it can handle (single byte) non-ASCII characters properly, provided that you supply it with a suitable `csf` file.

The `biblatex` package is also capable of handling conflicting encodings in `.tex` and `.bib` files, provided that you specify the encoding of the `.bib` file with the `bibencoding` package option. [4] For more details about some sample configurations you might try, please have a look at [4]. A good advice is to use `biblatex` and `biber`. [2]

One might then think about converting his `.bib` files to, say, `latin1`, and include them using

```
\begingroup
 \inputencoding{latin1}
 \bibliography{nameofthebibfile}
\endgroup
```

This will evidently work, but this is disadviced. If you are under Microsoft Windows ©, you will not need to use these four lines, simply because you have great chances to deal with `latin1` files. But, under Linux (UTF-8), your `.bib` files will be defaulted to `utf8`, which means that these lines would do the trick.

We reviewed how to correctly encode LATEX documents. We did neither speak about font encoding, nor about mapping multiple encodings into one. We assumed by 'encoding' the name 'input encoding.' We shall now take a quick look at these two subjects. The reader might also distinct these concepts from the language rules that need to be loaded appropriately for each language, using e.g.

```
\usepackage[language]{babel}
```

3.4 Font encodings

Roughly, *font encoding* defines at which position inside a TEX-font each letter is stored.

The default LaTeX font encoding is `OT1`, the encoding of the original Computer Modern TEX font. It contains only the 128 characters of the 7-bit ASCII character

set. [1] When accented characters are required, TEX creates them by combining a normal character with an accent. While the resulting output looks perfect, this approach stops the automatic hyphenation from working inside words containing accented characters. Besides, some of Latin letters could not be created by combining a normal character with an accent, to say nothing about letters of non-Latin alphabets, such as Greek or Cyrillic. To overcome these shortcomings, several 8-bit CM-like font sets were created. [9]

For more details, please check [9].

3.5 Mapping multiple encodings into one

Multiple input encodings could be mapped into one font encoding, which reduces number of required font sets. Font encodings are handled through the `fontenc` package:

```
\usepackage[encoding]{fontenc}
```

where `encoding` is the font encoding. It is possible to load several encodings simultaneously. [9]

3.6 Summary

The beginner might be impressed by the complexity of things. However, this is not that complicated for european and american languages. For these languages, you will generally use `utf8` as an `encoding` for `inputenc`. That should not pose any problem except for Greek, for which there are specific solutions.

Now that you will use `utf8`, your files' proper encoding should also be encoded with an encoding which will not be more restricted than `utf8`, so that you do not loose information. As `utf8` is one of the most complete encodings, the best thing is to save your files using `utf8`. It can be achieved under Microsoft Windows ©, and this is the default locale for Linux. If you are under Microsoft Windows ©, and that you do not encode your files using `utf8`, you might have troubles, but as, generally, people type documents in either their native language, or English, this should not be that problematic.

Keep in mind that we only saw an overview of these problems, and that specific languages might be problematic, especially for minorities. We did not give

a complete recipe to make everything work, but these guidelines should foster a better comprehension of this problem.

References

[1] *Surviving the TEX font encoding mess; Understanding the world of TEX fonts and mastering the basics of fontinst*, 1999. ftp://tug.ctan.org/tex-archive/fonts/utilities/fontinst/doc/talks/et99-font-tutorial.pdf.

[2] Fairbairns, Robin, Milde, Günter, Lehman, Philipp, Merciadri, Luca, Stephani, Philipp and Waßenhoven, Dominik. Encoding remarks (comp.text.tex discussion), 2010.

[3] Korpela, Jukka. A tutorial on character code issues, 2009. http://www.cs.tut.fi/~jkorpela/chars.html.

[4] Philipp Lehman. *The biblatex Package*, 2010. http://www.ctan.org/tex-archive/macros/latex/exptl/biblatex/doc/biblatex.pdf.

[5] Medina Arellano, Gonzalo and Merciadri, Luca. (Space between address, etc., and the beginning of the text: lettre class - comp.text.tex | Google Groups, 2010. http://groups.google.com/group/comp.text.tex/browse_thread/thread/048bbbf2d2537c39.

[6] Scott Pakin. Comprehensive LATEX Symbol list, 2008. http://www.ctan.org/tex-archive/info/symbols/comprehensive/symbols-a4.pdf.

[7] MacKichan Software. MacKichan Software - The Home of Sc. WorkPlace, Sc. Word, and Sc. Notebook, 2009. http://www.mackichan.com/index.html?techtalk/574.htm~mainFrame.

[8] Tuteurs Enseignement. (Écrire une lettre avec LaTeX), 2010. http://www.tuteurs.ens.fr/logiciels/latex/lettre.html.

[9] Wikipedia. LaTeX/Internationalization – Wikibooks, collection of open-content textbooks, 2010. http://en.wikibooks.org/wiki/LaTeX/Internationalization.