The PracTEX Journal, 2007, No. 3
Article revision 2007/08/17

Will TEX ever be *wysiwyg* or the pdf synchronization story

Jérôme Laurens

Email jlaurens@users.sourceforge.net
Website http://itexmac.sourceforge.net/pdfsync.html

Abstract Why editing Plain, LaTeX or ConTeXt documents can be such a pain ? Of course, we can use dedicated text editors and environments that are very handy, but we are far from the efficient graphical user interface of a modern word processor. In this article, we will point out some of the reasons why TeX has no *wysiwyg*[a]user interface and we will discuss the possible remedies. One of them is the pdf synchronization implemented in the `pdfsync` package. This technology will be explained, we will see its benefits and its limitations. Finally, we consider routes towards a better user experience. Here, we are mainly concerned with LATEX and pdf[b] output.

[a]What You See Is What You Get

[b]PDF (for Portable Document Format) is a technology invented by Adobe® in 1990 to create and share documents.

1 Time and memory are our enemies

When TEX was designed by D.E. Knuth some 30 years ago, personal computers were rather limited, their memory and CPU speed measured in kilobytes and megahertz. At that time, instantaneous typesetting was a nonsense unless we drop a great amount of both quality and efficiency. Indeed, TEX is well known for its very clever algorithms to break lines into paragraphs and paragraphs into pages, but these do need complicated calculations. Also, the macro feature is absolutely essential to manage sectioning commands, table of contents for example, but all this consumes lots of RAM. Long time TEX users will certainly remember the minutes spent to typeset just couples of pages, when patience was the rule. In fact, we can understand that to make the TEX program reasonably usable, things had to be dramatically optimized and there was no other choice than completely ignore graphical user interfaces (even if they were not yet popular).

Now that gigabytes and gigahertz are the standards, are we still limited by time and memory?

2 Typesetting can be extremely complex

Macro and page breaking illustrate how complicated can be the typesetting mechanism. It is obvious that changing a macro used throughout a TEX input file `foo.tex` involves changes throughout the output document (in general `foo.pdf`). In the same way, changing just a word can also have repercussions much more important than expected at first glance. If the size of the word changes, the lines might break differently, then the paragraphs, then the pages. The figures, tables, and more generally all the floating material, might also see their location modified. If the input file uses references to page numbers in the output document, then both the first and last pages can see their contents altered, and all the other pages consequently by some chain reaction. Finally, a slight change in the input file can lead to modifications spread over many pages of the output document, before and after what is changed.

This clearly shows that we cannot assume changes to be local, any modification is potentially global, so the only solution to be up to date is to typeset the whole input file from scratch.

3 A *wysiwyg* editor would increase complexity

Suppose you want to correct a misspelled word and turn an "a" into a "z", you just would like to select the "a" into the displayed output document then press the "z" key. How could a *wysiwyg* editor handle this task? First it would have to know exactly what in your input file generated this "a". There stands the first problem: the traceability of each character, each macro. Unfortunately, when TEX translates the input material into its own private internals, it does not remember the whole necessary information: the "a" character is turned into something called a node, the container file name and the file number being recorded for debugging purposes[1], but not the column number. Recording this last information would require more memory and also some tricky character index management.

1. This is the information used when, in case of error, you ask TEX to edit by entering the "e" character in the terminal.

But this is not the only problem: the traceability information should be remanent, which means that inside the pdf output document, any single character should keep track of the input file name and character index it was originated from. Not being a part of the natural pdf features, this would require dangerous acrobatics.

So far we reached a first conclusion: in order to have a *wysiwyg* TEX editor, one would need a huge amount of memory, speed, and engineering. We are constrained to reduce the ambitions.

4 Visual editors

Editors like LyX[2] and Scientific Word[3] provide us with an advanced graphical user interface, which resembles the output document for the main points. The sections are properly numbered and highlighted, the mathematical stuff is displayed with a dedicated font, colors are supported. All these visual effects have more meaning than the underlying command, and it is possible to edit text in place: this is a real benefit.

But these editors are not able to manage all TEX input files: for performance reasons, they only support a strict subset of TEX or LATEX. In general, this is not really annoying, but when it comes to advanced typesetting techniques, all the benefits are simply lost.

Another design is used by AUCTEX[4], the emacs extension dedicated to LATEX. It has a partial preview mode, where some critical parts of the input file are replaced by the real output. For example, displayed mathematical equations are replaced by the image obtained when typesetting only the equation. Of course, only the original input file is editable.

One can say that visual editors are just standard text editors with more or less extended capabilities. Let us explore now a more general concept to allow easy navigation between text input and pdf output.

2. "LyX is an advanced open source document processor running on many Unix and some non-Unix platforms" according to `http://www.lyx.org`.

3. A commercial document processor largely based on LATEX running on Windows™. See `http://www.mackichan.com/`.

4. AUCTEX is an extensible package for writing and formatting TEX files in GNU Emacs and XEmacs. See `http://www.gnu.org/software/auctex`.

5 The pdf synchronization

The first approach is due to Piero d'Ancona who had the idea of exploiting rather recent features of Han The Thanh's `pdftex`[5]. Whereas it was not possible to have complete traceability of all the characters, he thought that traceability of critical parts would be helpful, just like former `src-ltx` package did regarding DVI. Then, with the author, he could elaborate a LaTeX package, `pdfsync.sty`[6], that would automatically insert some properly designed control sequences in any input file, and create anchors in the output document for which the trace would be recorded.

More precisely, when using the `pdfsync` package in `foo.tex`, the `pdftex` engine will create a `foo.pdfsync` file containing both geometrical and traceability information. Let us see how it works on a concrete `foo.pdfsync` example:

```
foo
version 0
l 0 13
l 1 19
l 2 24
l 3 33
l 4 35
l 5 13
[...]
l 431 134
l 432 134
[...]
s 1
p 430 2368143 54651247
p 398 21086469 3154577
[...]
p 431 2368143 402063
l 433 134
[...]
l 464 173
[...]
l 527 215
```

5. `pdftex` is a concrete implementation of TeX with pdf output.
6. See `http://itexmac.sf.net/pdfsync` for more details and a complete set of specifications.

```
23   s 2
24   p 525 2368143 54651247
25   [...]
26   p 464 7149061 19993810
27   p 475 14470540 19043538
28   [...]
```

For direct (or forward) pdf synchronization, we start from a line number in `foo.tex`, say 173, and find where is the corresponding output. From the 20th line of `foo.pdfsync`, we can see that the anchor numbered 464 was registered for that line 173. Then from lines 26 and 23 of `foo.pdfsync`, it corresponds to a point of the second page of `foo.pdf` with TeX coordinates 7149061 19993810.

For reverse pdf synchronization, we can see that the point with TeX coordinates 2368143 402063 in the first page of `foo.pdf` was created with material from line 134, due to lines 13, 17 and 10 in `foo.pdfsync`.

To generate `foo.pdfsync` when typesetting, each time `pdftex` encounters maths, sections, boxes and other critical material, it appends to `foo.pdfsync` a "`l`" line like "`l 464 173`" where 464 is a unique anchor identifier and 173 is the current line number. When a new page is completely laid out, a line like "`s 2`" is appended to `foo.pdfsync`. It was necessary to manage the page indexing because page labeling or numbering need not be continuous in LaTeX. At that stage only, the geometrical information is available, so "`p`" lines like "`p 398 21086469 3154577`" are appended to `foo.pdfsync` for each previously recorded anchor. The package also has to properly manage included files to keep track of the container's file name, but this is not illustrated here.

Observing more precisely this example will convince even more that typesetting with TeX is a fairly complex mechanism. We can see that the "`l`" lines are written synchronously and increasingly, but the lines number they are refering to do not appear increasingly. This is most certainly due to macro extension which might not occur at parse time but sometimes later, at TeX's will. The "`p`" lines are not even written synchronously, just like if it were totally random. This is quite natural because all graphical objects in a laid out page need not be displayed in any particular order. We can also observe that sometimes no "`p`" line correspond to some "`l`" line, or multiple "`p`" lines do. Similarly, multiple "`l`" line can correspond to a unique location in a given page. Some temporary objects migh be created but never displayed, other might be used more than once. Thus pdf synchronization is not so simple, mainly because in case of doubt there are no criteria to make a good choice, this explains why the technology sometimes fails. However, people will appreciate the difficulties to design and implement the `pdfsync` feature.

The forthcoming limitations are not safe. The fact is that postponing write orders in `pdftex` until the page is completely laid out sometimes changes vertical spacing, paragraph or page breaking. Moreover, `pdfsync` macros can break existing packages. For these reasons, `pdfsync` should only be used during production stage.

Actually, only few tools are supporting `pdfsync` technology, with various levels of efficiency. AUCTEX is capable of forward synchronization towards `xpdf`[7], with an accuracy in the range of ± 1 page. In the Unix and Linux worlds, no other tool supports yet `pdfsync`, neither for direct nor reverse synchronization. On Mac OS X[8], two popular PDF viewers support reverse and forward synchronization, Skim[9] and iTEXMac2[10]. TEXShop[11] also have some kind support for synchronization as long as no external editor is involved.

6 The pdf synchronization by text extracts

When the Tiger version of OS X was released, developers could easily extract text from pdf documents and inspect their contents. Then TEXShop provided a synchronization method based on exact concordance. If the same sequence of characters is found both in the input and in the output, then it can be used to navigated between them. It has been chosen a sequence length of approximately 20 characters, less would lead do multiple choices and dilemnas, more would suppress concordances and reduce the efficiency.

Some people think that such a technology can be a replacement for `pdfsync`. They are mistaking: both should be used concurrently, this one for pure text and `pdfsync` for math and macros.

7. `xpdf` is a popular PDF viewer on Unix.
8. Apple System™ operating system based on Unix
9. See `http://skim-app.sourceforge.net/`.
10. iTEXMac2 has the most advanced implementation, and this is not due to the fact that the author is also iTEXMac2 developer. See `http://itexmac.sf.net`
11. The most popular TEX environment on Mac OS X. See `http://www.uoregon.edu/~koch/texshop`.

7 The pdf synchronization by words

To improve even more both accuracy and efficiency, a new approach is also using the file contents to synchronize. It is based on a simple observation: the input file contents and text extracted from the pdf output are not so far from each other. The main differences concern macros, line breaks and math, but the words remain essentially the same.

Thus, how can we achieve synchronization ? First of all, one cannot use the `diff` utility as is due to performance reasons, but the idea can help. Given a word in `foo.tex`, we find the two preceeding words and the two following words, wiping out macros, line breaks, and maths, but taking accents into account. Then we find in the pdf output all the text extracts that contain these five words in exactly the same order, starting on the first and ending on the last. In case different extracts are found, only the shortest ones are kept. If there is still a choice to be made, the `pdfsync` technology helps. This is extremely efficient for direct synchronization, and works similarly from output to input.

This is the most accurate and efficient method for pdf synchronization, in both directions, and iTEXMac2 is the only application implementing it. Instead of asking to display the point for a given line in a given input file, you just ask to display the point for a given character in a given text extract of a given input file. Then, iTEXMac2 will parse the words and take care of everything to properly highlight the corresponding character in the pdf document.

8 Can't we do better?

Concerning `pdfsync`, the author already received significant contributions from very qualified people, mainly Hans Hagen and David Kastrup, in order to publish the package on CTAN. But the whole technology should definitely be embedded directly in the `pdftex` engine and the end user should not have to worry about it at all. This would definitely prevent polluting the pdf output with erroneous vertical spacing and any other undesirable or even fatal side effects. Moreover, this would make `pdfsync` available indifferently for TEX, LATEX, ConTEXt and any other format.

The work is in progress and will certainly be complete in a few months. There is not great risk in guessing that the tools on Mac OS X will quickly be adapted to this new kind of synchronization, but the same does not hold under Unix and Linux.

More generally, it is actually quite impossible to achieve the ultimate goal of complete *wysiwyg*. If extending TEX internals to really support character traceability is certainly feasible, TEX as a typesetting engine remains of an unbearable slowness. For that point, improvements would possibly come from incremental typesetting: when TEX processes an input file, it should remember what is has done before and modify only what really needs to be modified. This could be done by splitting TEX into a macro manager and a typesetting server. As a consequence, one would also have to separate typesetting macros that drive the layout from structuring ones like `\section{...}`.

All this implies radical changes in TEX internal architecture. Is it feasible? Only the future will tell us.