The TEX Wrapper Structure: a basic TEX document model implemented in iTEXMac

Jérôme Laurens
Université de Bourgogne

February 25, 2005

1 Introduction

This presentation primarily concerns the high level user interface of the TEX typesetting system. In general, people find it difficult to work with TEX due to the powerful syntax, numerous auxiliary files created or managed, and the user interface that has very little in common with standard word processors. Moreover, sharing TEX documents with colleagues is often delicate as soon as some non standard LATEX is involved or, more frequently, there are some significant differences in the computer configurations. The purpose of this article is to lay the foundation for the TEX Wrapper Structure, which aims to help the user solve this kind of problems.

We first explain what could be the desiderata for a TEX document object model, then we give a precise description of the TEX Wrapper Structure, discussing the various solutions and the final choice. Finally, the concrete implementation used by iTEXMac[1] demonstrates an example of user interface.

An appendix briefly presents the latest developments concerning PDF synchronization which is a MacOS X specific feature of great interest for the whole TEX community.

2 A TEX Document Model

2.1 *De facto* document model

A document model aims to describe the storage and use of a certain kind of data: a simple document model might be a linear text, which is an ordered list of 8 bit numbers following the ASCII rules and stored in one flat file. More complex document structures are used either to describe data contents, for example Adobe's Portable Document Format, or to store them, for example old MacOS operating sytemes use a hierearchical file system with resource forks to allow file to store structured data. Regarding these two points among others, TEX is very specific mainly because it does not pose *a priori* any document model, letting the end user use its own *de facto* model. The question is to identify what core structure should have a TEX document model, that should be shared by quite all documents including the ones already existing.

Actually, a *self contained TEX document* is a series of files gathering data as various as images, linear text, formatted text, macro packages (LATEX style), code libraries (libjpeg...), engines (TEX, MetaPost) and their calling options. Of course this makes really huge documents, such that common parts are naturally eliminated, hoping that they will be available everywhere and every time one will ever need them. This results in some kind of *weak TEX document model* which has proved to be efficient, except in some rare situations where the syntax was broken by some package update, and less rare ones where engine options have been forgotten... Far-sighted TEX users carefully keep the various log files coming from typesetting because of the versioning information they contain. It is extremely helpful when fixing update problems, but still relies on non negligible human expertise where one could reasonably expect full computer assistance. When a strategy is available to record version information, it will be added to the TEX Wrapper Structure.

Generally speaking, a TEX document is composed of different kind of graphical objects, from linear text to pictures, possibly splitted into different files. There is no real problem concerning the various graphical data

[1]iTEXMac, one of the open source TEX front-ends on MacOS X, was presented during EuroTEX 2003 and TUG2004. Further information at `http://itexmac.sourceforge.net`

formats but the same does not hold for TEX source files. Any TEX user knows that a source document is not correct as long as it has not successfully passed TEX digestive process. More experienced users are perfectly aware of the problems that can appear when using certain combinations of macro packages. All this makes the data part of a TEX Document Model very difficult to define *a priori* in a complete and explicit description. This design, being as open as possible, is a real advantage because it provides quite unlimited document types. But at the same time, it does not take into account the document preparation stage and does not provide any help to the user in his real life struggle for document elaboration.

2.2 The meta information

For that purpose, advanced TEX dedicated editors have designed their proper strategy to assist the user with extraneous information not really necessary but missing when absent: the meta information. For example syntax attributes highlighting (marking TEX tokens, comments and other stuff with special colors) is a clever use of the information actually available as is in a source file. This can be improved by some syntax checking, that could mark bad commands just like the spell checker marks the misspelled words. For this to work efficiently in a real time context, we must collect the macros defined in the context and cache the whole dictionary list to improve access. Similarly, parsing the document contents for sectioning commands provides the user with a map that improves the overall sight and the navigation inside the document. All this is more or less filtering or interpreting the existing information to make it more accessible. Moreover, editors are free to add their own information if they think it is relevant. We can see that in fact, real TEX users may need more information that actually available in a TEX document, and TEX does not care about this kind of meta information. The TEX Wrapper Structure will mainly consider this point.

2.3 The document storage

As we must preserve actual TEX documents in a backward compatibility issue, we are only concerned with the document storage, more precisely the location where the different files are stored. Some of them must be located in definite folders, according to the TEX environment (in general following the TEX Directory Structure rules), while the user is absolutely free to name others. For them, some weak naming rules could help in their organization, without limiting their use. For example, people generally gather their graphic files in folders named images, graphics, pictures or whatsoever but there is not yet a widely spread strategy to become part of a TEX document model. Moreover, we must admit the use of different naming strategies to best fit the numerous situations one can imagine, for example, a unique image directory is certainly not advisable when the document is expected to contain thousands of logically organized images. Finally, the only naming rules we can safely state concerns the meta information and will be addressed by the TEX Wrapper Structure.

3 The TEX Wrapper Structure

We define the core TEX Wrapper Structure gathering information useful to any editor or utility, then we detail the TEX project concept and we briefly describe the concrete implementation developed in iTEXMac. This is a weak TEX document model given through a series of compliance rules, only assuming an underlying hierarchical file system. We also assume that all the document files but the standard macro packages are collected in one enclosing directory, but *a priori* different documents can share the same directory.

3.1 The core TEX Wrapper Structure

The only purpose of the core structure is to separate the document data, which is necessary, from the meta information, which is supplemental. Actually, the meta information is stored either in the very TEX source file (for example the %& first line trick to code for the format, the first commented line for TEXEXEC, emacs local variables to code for the string encoding, AUCTEX local variables in the file trailer), or an external file (the `.aux` LATEX file, the `Auto/` directory where AUCTEX caches its style attributes, the TEXniCenter projects) and each tool defines its own strategy without really taking care of one another. It is not yet the point to define a unique and complete set of meta information, but we are concerned with the storage location of the meta information. For practical reasons, it appears that some information such like the string encoding and the language should live near the document they are referring to, but other information including the list of project files and the root document identifier should live in a shared data base. If we consider all the tools of the TEX typesetting system, the simplest solution from the user point of view, is to collect the whole meta information into one central dedicated location. That way, no more meaningful comment will pollute the TEX source thus preserving the meta information from hazardous manipulations and preventing an innocuous TEX comment to become suddenly active while a utility has silently put some implicit information in it.

Finally, I strongly recommend not to use TEX comments anymore for anything else that commenting, except when conforming to a publicly available and widely accepted syntax rule.

3.2 The TEX project paradigm

With emacs' AUCTEX mode and TEXniCenter we already mentioned, the old DirectTEX pro is another example of editor that stores meta information in an external file. We propose to collect this information in one dedicated directory. So, a TEX project is just a directory named *document*.`texp` (“`texp`" stands for TEX Project) where shared or private meta information should be stored. The possible interference with already existing TEX documents is quite void because the `texp` extension is not yet used, this ensures a full backward compatibility.

To define the mapping linking projects to files, the TEX project is expected to maintain a list, either explicit or implicit, of all the files it is meant to manage. But conversely, it is not strictly necessary for the TEX source files to know the project they belong to (as for AUCTEX) because this information can be retrieved easily if we impose that TEX projects only manage files at the same level of below themselves in the file system hierarchy. Then, given a file path, we just have to scan the file hierarchy up to the root for TEX projects and only keep the appropriate ones.

The contents of the TEX project directory *document*.`texp` is described in the sequel. The user is not expected to view nor edit this data, so the format primarily concerns the programmers. More precisely, it is a balance between a flat XML file and an atomic directory structure, both suitable for information hierarchically organized. The “/" character is used as path separator.

Key	Class	Contents
`isa`	String	Required with value: `info`
`version`	Number	Not yet used but reserved
`files`	Dictionary	The paths of the files involved in the project wrapped in a `files` dictionary described in table 2. It is an indirection table suitable for file name management. Optional.
`properties`	Dictionary	Attributes of the above files wrapped in a `properties` dictionary described in table 3, this is were string encoding and spelling key are recorded. Optional.
`main`	String	The *fileKey* of the main file, if relevant, where *fileKey* is one of the keys of the `files` dictionary. The main file is the one to be typeset or processed. Optional.

Table 1: `info` dictionary description where the TEX project maintains the list of known files, their properties and the main file identifier.

- *document*.`texp/Info.plist` is an XML property list for a general purpose meta information wrapped in an `info` dictionary described in table 1 and subsequent tables. This is optional.

  We make use of the XML property list data format storage as publicly available at

  `http://www.apple.com/DTDs/PropertyList-1.0.dtd`

  It is indeed MacOS X centric but two PERL modules are available on CPAN to parse such XML files: Mac-PropertyList[2] andMac-PropertyListFilter[3]. Moreover, this can be changed in forthcoming versions without causing any harm from the user point of view.

[2]`http://search.cpan.org/~bdfoy/Mac-PropertyList-0.9/`
[3]`http://search.cpan.org/~jgoff/Mac-PropertyListFilter-0.02/`

Key	Class	Contents
fileKey	String	The path of the file identified by the string *fileKey*, relative to the directory containing the TEX project. Each file key is unique. While the file name is subject to changes, the file key will never change: the latter is a strongly reliable file identifier. In general, no two different keys should correspond to the same path.

Table 2: `files` dictionary description: an indirection table particularly suitable for file name management.

Key	Class	Contents
fileKey	Dictionary	Language, encoding, spelling information and other attributes wrapped in an `attributes` dictionary described in table 4. *fileKey* is one of the keys of the `files` dictionary.

Table 3: `properties` dictionary description: to each key identifying a file is associated a dictionary of attributes.

Key	Class	Contents
`isa`	String	Required with value: `attributes`
`version`	Number	Not yet used but reserved
`language`	String	According to latest ISO 639. Optional.
`codeset`	String	According to ISO 3166 and the IANA Assigned Character Set Names. If absent the standard C++ locale library module is used to retrieve the `codeset` from the `language`. Optional.
`eol`	String	When non void and consistent, the string used as end of line marker. Optional.
`spelling`	String	One of the *spellingKeys* meaning that the property list at *document*`.texp/`*spellingKeys*`.spelling` contains the list of known words of the present file wrapped in a spelling dictionary described in table 5. Optional.

Table 4: `attributes` dictionary description

Key	Class	Contents
`isa`	String	Required with value: `spelling`
`version`	Number	Not yet used but reserved
`words`	Array	The array of known words

Table 5: `spelling` dictionary description for the list of known words.

- *document*`.texp/frontends` A directory dedicated to front-ends where they store private meta information.
- *document*`.texp/frontends/`*name* A private file or directory dedicated to the front-end identified by *name*. The further contents definition is left under the front-end responsibility. The directory at

document`.texp/frontends/iTeXMac`

is reserved for iTEXMac private use, maybe AucTEX can move its `Auto/` directory into

document.`texp/frontends/AucTeX`

and TEXniCenter can use

document.`texp/frontends/TeXniCenter`.

This cooperative design is based on a strong separation of private meta informations from each other front-end, it prevents corruption and allows better recovery in case of error. Moreover, synchronization problems that may appear when two different utilities access the same flat file do not occur.

- *document*.`texp/users` is a directory dedicated to users and should not contain any front-end specific data. This is optional and reserved for further user.
- *document*.`texp/users/`*name* is a directory dedicated to the user identified by *name* (not its login name). Not yet defined, but private and preferably crypted.
- *document*.`texp/`*spellingKey*.`spelling` is an XML property list for lists of known words wrapped in a `spelling` dictionary defined in table 5 and uniquely identified by *spellingKey*. This format is stronger than a simple comma separated list of words. This is optional.

  We assume that a text document is multilingual and can have different spelling contexts, all of them being defined by a language with a dictionary and a list of known words. At this time, MacOS X programming interface does not allow to have more than one spelling context per open file, and the same might hold for other operating systems. So, each file is expected to have only one spelling context defined by a language and a spelling key, both defined in the `properties` dictionary (see the description in table 3). Then, a multilingual document will be splitted into files according to the language and the list of known words.

  Notice that there is no pre definite correlation between a language and a list of known words. And this design is certainly not the best we can elaborate, but it appears to be sufficiently efficient.

3.3 The TEX Wrapper Structure implemented in iTEXMac

The graphical user interface developed in iTEXMac takes benefit of the TEX Wrapper Structure. Private informations are cached to improve the user experience: window size and positions recording are the classical examples. Also, meta information about the engine and options used to typeset the document are stored, they are used to launch the appropriate utility with appropriate arguments assuming a teTEX like distribution is available. This should be shared once the latest TEX live is well established.

Technically, iTEXMac uses a set of private, built-in shell scripts to typeset documents. If this is not suitable, customized ones are used instead, possibly on a per document basis, but no warning is given then. No security problem has been reported yet, most certainly because such documents are not shared.

Notice that iTEXMac declares both `texp` and `texd` as document wrapper extensions to MacOS X, which means that *document*`.texp` and *document*`.texd` folders are seen by other applications just like other single file documents, their contents being hidden at first glance. Using another file extension for the TEX document will prevent this MacOS X feature without losing the benefit of the TEX Wrapper Structure and its TEX project.

4 Appendix: The **pdfsync** Feature

During the document preparation using the TEX typesetting system, the correspondence between the output and the original description code in the input is of frequent use, unfortunately it is not straightforward. Some commercial TEX frontends (Visual TEX[4] and TEXtures[5]) introduced a workaround. Then LATEX users could access the same features with a less-efficient implementation through the use of `srcltx.sty`, which added source specials in the DVI file. The command line option `-src-specials` now delegates that task to the TEX typesetting engine.

iTEXMac fully supports this synchronization allowing to jump from the DVI file to the `.tex` source and back. Moreover, Piero d'Ancona and the author have extended this feature from the `.tex` to the `.pdf` output.

[4]`http://www.micropress-inc.com/`
[5]`http://www.bluesky.com/`

While typesetting a *document*.tex file with LATEX for example, the pdfsync package writes extra geometry information in an auxiliary file named *document*.pdfsync, subsequently used by the front ends to link line numbers in source documents with locations in pages of output PDF documents. iTEXMac, TEXShop[6] and TEXniscope[7] both support pdfsync.

The official pdfsync web site where file specifications and more complete explanations will be found at:

http://iTeXMac.sourceforge.net/pdfsync.html

Unfortunately, the various pdfsync files for Plain, LATEX or ConTEXt are not completely safe. Some compatibility problems with existing macro packages may occur. Moreover, sometimes pdfsync actually influences the final layout; in a case like that, it should only be used in the document preparation stage.

Notice that the pdfsync approach is different from Heiko Oberdiek's vpe.sty.

[6]http://www.uoregon.edu/~koch/texshop

[7]http://docenti.ing.unipi.it/~d9615/homepage/mac.html