EuroTEX 2001

Directions for the TEXLive system*

Fabrice Popineau

Abstract

This paper is about the current status of the TEXLive software.

The first part of the paper will address the structured description of its content and how the Windows [1] setup program can use it. The past experiments with the Windows installer have revealed that the problem was harder than expected. The new TEXLive 6 description files will allow a more effective way to use the setup program. Some further enhancements are even scheduled.

The second part of the paper will address a set of possible extensions to the Web2C/Kpathsea pair (read it as a call for code contributions!). Some aspects of its use were not foreseen when it was devised and it may be time for an enhancement.

1. Introduction

TEX is a great piece of software, but it used to be not so easy to distribute. As time passed, people arranged so called *TEX distributions* for various platforms, which became easier and easier to install and use, even for people not really acquainted with TEX.

To name only a few, there was emTEX for DOS platforms, which came compiled, as usually people do not have development tools for this platform. There was also Web2C for Unix, which came in source form, because all Unixes have development tools. This concerns the programs. On the other side, the collection of TEX macros available around the Internet did not stop to inflate over the years.

All these distributions matured and Web2C turned out to be really portable so that its programs were made available on all major platforms. The most successful instantiation of Web2C as a standalone distribution was the great teTEX distribution assembled by Thomas Esser. At the same time, the TEX Directory Structure was standardized, and the layout of the support files for the programs almost frozen. It was time then to build the first multiplatform ready-to-use distribution: the TEXLive project was born.

2. The 6th edition of the TEXLive system

The TEXLive system gathers almost all *free* TEX related software available from the CTAN repository. By this I mean all the programs directly related to TEX and its use (around 240 for Unix, a few more for Windows) and most of the free macro packages available on CTAN plus documentation. This is around 40000 files on a CD-ROM. So there is one big problem: how to manage such a huge pile of files?

We can split it in two parts. First, CTAN is great because it holds all TEX related files, it is managed, and you can find whatever you need on it if it does exist. But the files there are not directly usable: you need to unpack, compile, install at the right location on your site. Not something all people are ready to do or even not easy to do at all. Anyway, CTAN aims at being a repository for contributions to the TEX world, no more no less.

So in a sense the TEXLive system can be seen as a projection of CTAN onto some instance of the TDS with all files ready to use. This task has been almost completely automated for the macro packages by Sebastian Rahtz, who wrote a set of Perl scripts that can automatically convert any of these packages from their CTAN form into their usable form. That means that the archives are unpacked, the `.dtx` files are processed, documentation compiled, files moved to the right TEXLive location and so on.

The second problem is to find a way to present the user with those files in an understandable way. So we need to build a structured description of the files.

2.1. Structured description of the TEXLive system

The RDF format has been chosen for describing the components. RDF is an XML application that allows one to express sentences of the OAV (*Object – Attribute – Value*) kind. One can use it to claim that such url on the Internet has an attribute of such value. This model is powerful enough to

* Thanks to Sebastian Rahtz for putting up the TEXLive project.

[1] Later in this paper, all the various Windows platforms will be designated by the generic Win32 acronym.

express complex sentences, but we don't need the advanced features for the moment. However, relying on this language is a best bet for the future when we might need them. We have chosen a dedicated namespace for the so-called TPM (*TEX Package Management*) description files.

This edition introduces a new structured description of the various components. The current description is based on collections and packages. What is a collection and what is a package? A collection is intended to be a *collection of packages*. Each of these components has a common set of attributes which are given below for the tex-basic collection:

```
<!DOCTYPE rdf:RDF
  SYSTEM "../tpm.dtd">
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:TPM="http://texlive.dante.de/">
   <rdf:Description about="http://texlive.dante">
      <TPM:Name>tex-basic</TPM:Name>
      <TPM:Date>2001/03/16</TPM:Date>
      <TPM:Version>1.0</TPM:Version>
      <TPM:Creator>rahtz</TPM:Creator>
      <TPM:Title>Essential programs and files
    </TPM:Title>
      <TPM:Description>
      These files are regarded as basic for any
      TeX system, covering plain TeX macros,
      Computer Modern fonts, and configuration
      for common drivers.
      </TPM:Description>
      <TPM:Flags default="yes"/>
      <TPM:RunFiles size="14957">
texmf/tex/generic/hyphen/hypht1.tex
...
texmf/tpm/collections/tex-basic.tpm
  </TPM:RunFiles>
  <TPM:BinFiles arch="i386-linux" size="4706207">
bin/i386-linux/texdoctk
...
bin/i386-linux/MakeTeXPK
</TPM:BinFiles>
      <TPM:BinFiles arch="win32" size="4194277">
</TPM:BinFiles>
      <TPM:Requires>
         <TPM:Package name="bibtex"/>
         <TPM:Package name="bluesky"/>
...
         <TPM:Package name="texlive"/>
         <TPM:Package name="tex-ps"/>
         <TPM:Package name="bibtex8bit"/>
      </TPM:Requires>
      <TPM:RequiredBy/>
   </rdf:Description>
</rdf:RDF>
```

Information about the components can be divided into three parts:

1. Information about the component: name, date, version, description, and so on. There is a special Flags tag, which is used to give directives to the setup program. For the moment, it is used to specify that some component should be selected by default, or that it is only remotely available.
2. Files lists, split into:
   bin files which are tagged by the target architecture given as an attribute and which specify all the files that should go into an architecture dependent directory (namely the programs);
   run files which refer to the files used by the TEX system and which are architecture independent, i.e. mostly macro files;
   source files which refer to the source files for the macro packages; the source files for the programs are not yet indexed this way;
   doc files which refer to the documentation files that come with the component;
   remote files which are used only for some Win32 support files that are not free or for which there is no space on the CD-ROM, hence that are only remotely available.
3. Dependencies: the other required components needed to make this component work.

This information is sufficient to build a simple management system with the following features:

Installation install the files for the first time on the computer;

Maintenance allow the user to add, upgrade or remove components;

Removal remove all the files that have been installed.

All these operations should be done with consistency ensured. By this I mean that all the required packages are installed whenever you choose one, and that when you upgrade some package, all the required packages are upgraded if needed. Obviously, the database of TPM files needs to be sound by itself and should reflect the reality.

Enforcing consistency can be tricky. Dependencies define a graph between the components and the job of the management program when you ask it to install something is to find a minimum

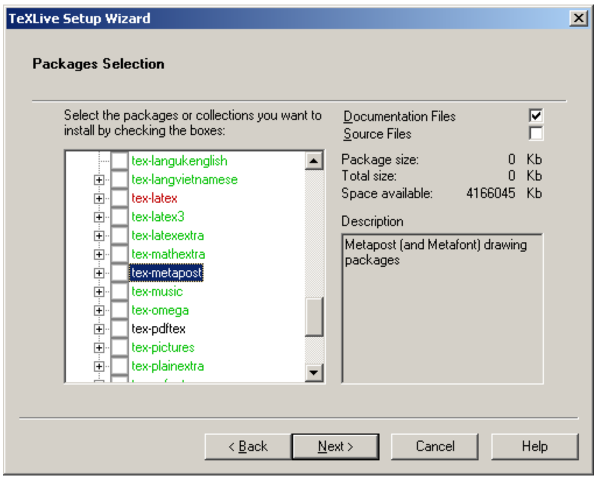

Figure 1: Package selection page of the TeXSetup wizard.

spanning tree over this graph that covers all the selected packages. Using such a system where consistency is enforced, it may happen that you will try and be denied to deselect some component: all dependencies may not be obvious to the user and it is rather intriguing to click on some checked box, and to fail to unckeck it! Maybe in this case the system should display the reason why the component is required. So dependencies and package bounds should be carefully designed.

Experiments have been done with the Windows TeXSetup program. The version in the current TEXLive system allows the user to install, to add and to upgrade packages. Figure 1 shows the screen when the user wants to add or upgrade some of his packages. The black labels (only `tex-pdftex` on the figure) indicate that the component is installed and up to date, the red ones (`tex-latex` on the figure) that the component is out of date and the green ones that it is not yet installed (all the other packages on the figure).

2.2. How to go further in describing the TEXLive system?

The system is working properly at this point. However, not all issues have been adressed.

There is a small step that will be taken quickly[2]. Currently, packages as well as collections hold files. This should not be the case anymore. All files should be moved into packages, and collections shoud only hold a set of requirements. This will not prevent packages from requiring collections if that is needed. It will only ease the visualisation of dependencies.

In many cases where collections hold files, the files are binaries. So the binaries will also be split into packages corresponding more or less to the various parts in the program sources of the TEXLive system. So in this new description, the `tex-omega` collection will require some `omega-basic` collection and some `omega-lambda` collection, the `omega-basic` collection requiring the `omega-binaries`, `omega-base` and `omega-fonts` packages. Further, this will enable us to introduce a dependency between the `omega-binaries` package and the `kpathsea` package. You may wonder why it is needed. Under Windows, Kpathsea is distributed as a DLL as are some other parts of the system, so if you upgrade some program depending on Kpathsea, you potentially need to upgrade Kpathsea itself, and hence all other programs depending on it. But once this dependency is explicit, even upgrading the programs will be done in a safe way.

Another point that will probably be adressed is the removal of the division into doc-files, source-files and run-files. The problem is that at the moment you can choose to install each package with or without documentation and with or without sources. This has two flaws:

1. Each selected package is flagged as being installed, but nothing is remembered about its documentation or source files. The 'no-doc' flag is global, but the doc-files are defined per package. This is not so much of a problem as long as you can reinstall each package alone and choose to add its doc and source files if needed, but it is not very consistent because when you upgrade the package, you do not know if you should do it with or without doc and source files.
2. It is worse for some packages that hold only documentation, because they can be flagged as installed while only their `.tpm` file got installed (with the 'no-doc' option selected globally).

Given that documentation is very useful for many packages[3], the documentation files will be combined with the run files. On the other hand, the source files, which are usually useful only to a

[2] Maybe even achieved by the time of the conference.

[3] And that the cost of disk-space has dramatically dropped!

few people, will be described in a separate set of .tpm files, and there will be a separate collection (resp. package) for the source files of each collection (resp. package). This set of source files related collections and packages will form a separate tree from which users can select components. The setup program can have an option to automatically link any selected package to its source counterpart so that sources get installed easier.

Now, if we look further, we can wonder if a different, much bigger step should not be taken. We said that the TEXLive system is the projection of the CTAN content onto a ready-to-use axis. But CTAN already has its own set of description files, namely the Graham Williams Catalogue! Which incidentally is on the TEXLive CD-ROM. So we have two sets of description files. In fact, there is a third useful description of the components, which comes with the texdoctk program[4].

So we might think that three is two too many. One description should be enough. It is true that a merge between the Catalogue and the description files of the TEXLive system is highly desirable. But this is a huge work to undertake, and it could be done only by setting up a new procedure to submit files to CTAN. In this procedure package authors would be asked to build the XML description file for their package and also probably to follow very strict rules to package their contribution. Maybe this procedure can be automated by providing people with the right set of scripts, but this needs to be investigated.

2.3. Remaining problems about the setup system

The configuration files are not yet handled very nicely. The problem has been solved locally for the language.dat file. It has been split into several parts one line long, each one assigned to some collection specific to this language. When the user selects his collections, all the parts are assembled and the specific language.dat is build.

This process could be generalized to some other configuration files: fmtutil.cnf, map files in pdftex.cfg and config.* for dvips and gsftopk. However, doing this is hardcoded for the moment. It would be better to design and standardize some mechanism that would allow one to specify how the configuration files are built through rules and regular expressions.

Another remaining problem is about the setup programs. The Unix install-cd.sh script shell does not actually use the RDF files, but some preprocessed version of them. It also has no graphical user interface.

The Windows installer is a regular MFC dialog-box based application with a specific behaviour: a wizard in the Windows world. It relies heavily on platform specific features (dialog items like tooltips and tree control, Internet functions) so its code is not at all portable to the Unix world.

There is not yet any installer for the binaries on MacOS/X, and there is little chance that if one is written it could even be reused on another Unix system.

Reasoning in the context of a multiplatorm distribution, it would be nice to have a portable installer. However, requirements are very different between Unix and Windows. The other problem is that the installer needs to be standalone, and none of the tools that would allow one to quickly build a portable installler like Perl or Python, have the feature of building standalone programs. If this feature is ever introduced for all platforms, then this is clearly the way to go. But we can't expect that on every platform, users will have a complete Perl/Tk or wxPython installation, with the right GUI libraries, right version of the files and so on.

The other way of thinking, probably much more realistic, is to rely on a specific setup system for each platform. Sebastian Rahtz is willing to provide RPM packages for Debian and RedHat Linux users, and in fact he already did it as an experiment. Not all details have yet been sorted about – especially related to configuration files – but that's on the way for the next release.

Now that Microsoft has provided its own installer for all the versions of Windows, it would probably be better to switch to it[5]. It was not obvious that the first version of this system would be usable for the TEXLive system mainly because of the large number of files, but this Windows Installer system has quickly evolved and can't be ignored today.

[4] A nice Perl/Tk script that points you to the documentation inside a teTEX installation.

[5] The latest experiments about permissions under Win2K would argue for this solution.

So the common starting point will be the RDF description of the packages, and platform dependent versions will be built by scripts for each kind of setup system.

2.4. Other enhancements

We would like to get even more granularity. For instance, the "langgreek" collection has about five different Greek setups. Which one is best? In general, it is time we started *to cut down* on LaTeX packages. Perhaps ConTeXt shows us the way forward.

The new package description allows us to define different distributions for different uses. Most people do not want the whole set of programs. We would like to define genuinely small TeX distributions, as small as possible. It could help to make TeX more popular. Setting up a pdfLaTeX based distribution, without the METAFONT system but only Type1 fonts, for Western European countries, and a reasonable amount of LaTeX packages takes less than 30Mb. Maybe it is still too much, but this is way less than the default setup of the previous TeXLive versions.

3. At the heart of the system

There are other parts of the system that could be enhanced to gain greater ease of use.

3.1. Unforeseen usage of Web2C

The `\write18` primitive was very successful among advanced TeX users. Using this primitive, you can make TeX execute commands while compiling your document, but you can also make these commands write their result into a file and `\immediately` read it! This is very interesting because it means you can make TeX communicate with other programs bi-directionally.

So what is the issue with an extensive usage of this command? The main problem is that the Kpathsea library has not been designed to be reentrant, and that none of the programs that are chained share any of the data structures used by Kpathsea. So the various hash-tables are built several times, which takes time, especially if you consider building the hash-table for the `ls-R` of the TeXLive system, which may hold all the files.

If we consider the Win32 platform, the situation is even worse under NT/Win2K and the NTFS filesystem because of the underlying Unicode filenames that need to be compared case-insensitively. It appears that the conversion towards uppercase names is a little bit slower under NTFS. Benchmarks on a PII-400Mhz machine show that building the hash table takes up between 2 s and 3 s without the conversion to uppercase names, and around 1 s more with it.

When I was told some people could call METAPOST hundreds of times from their pdfTeX run, each METAPOST run calling in turn pdfTeX, I quickly hacked Kpathsea to put the hash tables in shared memory, and save the initialisation time. The result was as expected. Processing the following file:

```
\input supp-pdf.tex
\pdfoutput=1

\immediate\openout10=fabrice.mp
\immediate\write10{beginfig(1);fill fullcircle
    scaled 4cm;endfig;end.}
\immediate\closeout10

\loop
  \immediate\write18{mpost fabrice.mp}
  \convertMPtoPDF{fabrice.1}{1}{1}\vfill\eject
  \ifnum\count0<100 \repeat

\end
```

with a debug version of kpathsea and the hacked version, time dropped from 2min 30s to 0min 30s. So there is indeed some potential gain for huge jobs there. Obviously, the final gain depends mostly on how much time is needed to do the typsetting itself.

However I think that, if we want to make the design of the whole system evolve, such a step will be needed. Kpathsea will need to act as a server. At what level needs to be discussed (probably a DCOM server on Windows, but read further about a plug and play TeX system).

3.2. Extending Kpathsea towards the Internet

Kpathsea is a nice library which has provided a common interface to *paths* (ways to access some file) across different platforms and architectures. It is not very strange that today people wonder if it could be extended towards the Internet. It would allow you to include files in your documents using urls. We could even define it as a new protocol and use it like this:

```
\RequirePackage{tds:graphics.sty}
...
\font\foo=http://www.font.net/foo/bar/x
```

In fact, there is nothing preventing us from doing so. Some experiments along this line are on the way under Windows. The TeXSetup program required the ability to download files, so the basic functions to grab a url to a local file or to a string are there. It is not that hard to make kpathsea aware of urls as a kind of file. And it is another good test of the robustness of its API. The only problem is that downloading files takes time, and it is subject to failure because of dead connections and so on. So adding this feature should be done carefully:

- Whenever a file is downloaded, display some progression of the download so that users can decide to kill the job if it is stuck or too slow.
- Add a file caching system to avoid downloading the same file twice if it is used twice by the same job. This cache system should also be given a maximum amount of disk space to use and an LRU mechanism to decide which files to drop when it needs to reuse the space. One could also argue that this is the job of a proxy server, but not everybody can set up one. So we might be forced to implement a simple one.

These are the minimum requirements of such a feature.

3.3. Other possible extensions

Memory allocation should be made dynamic throughout the Web2C system. It is really annoying when you have to rebuild your format file because you have reached the current limits. And there is nothing in your document to specify that you used an enlarged TEX to compile it, so beware if you give the file to someone else. Moreover, almost half of the `texmf.cnf` file is dedicated to specifying those limits. Worse, not all the arrays used in those programs have been made extensible, but only the ones that are most likely to require it. However, every year, we need to add arrays to the list of extensible ones per users' requests.

Kpathsea should make it simpler to put files in the TDS tree. Currently, it is done by a very complex set of shell scripts. I know that some people advocate for shell scripts being easy to use and read but I'm not sure it is the vast majority. Being able to provide your own function to change the behaviour of Kpathsea, or to configure the way it will store files, why not. But in any case, Kpathsea should be complete and provide a decent, canonical and simple system to store the generated files. This is also a requirement for the point discussed further about a better integrated TEX system.

4. Core code considerations

4.1. Sharing and reusing code

Various routines are duplicated in several places, or implemented in different ways. These routines deal with very different things:

- Basic ones, like reading a line in a text file. If we want to support several platforms with different end-of-line conventions, then we should take care of the problem, preferably in only one place. It appears that for example, BibTEX does not use the same routine to perform this function as the one used by TEX. The TEX routine has been enhanced to support different eol conventions, but the BibTEX one has not, so we eventually run into the buffer size problem with BibTEX when it is fed with a MacIntosh file.
- Higher level ones, like reading a DVI file. This is done several times by very different tools for different needs (xdvi may not have the same requirements as dvitype), but all in all, the feature is the same: being able to parse a DVI file and access all the objects in it. Maybe the DVIlib by Hirotsugu Kakugawa presented last year at TUG2000 could be a starting point.

Any project aiming at enhancing the current set of programs should have a close look at these redundancies, try to implement some kind of lowest common multiple set of features in terms of libraries, then make the programs use these libraries and remove the original code. It has been done once with great success: Kpathsea is the way to go. It can be done further. For example, there is the VFLib project, which could be used to let all the programs share several parts of code related to font handling.

4.2. Towards an integrated plug and play TEXsystem.

What is the aim of this discussion? We love TEX and we would like it to be more popular and shared by many more people. So what do we need to make it easier to use and install? Basic computer users are used to programs with a nice interface, preferably with lots of menus and buttons, they are not used to dealing with dozens of console mode programs. In this respect, any TEX system looks old.

So what do we have to do to turn TEX into something more appealing? We have for example XEmacs (or maybe Emacs, but I don't know much of the latest developments) which has a very nice design that could be made to embed TEX. How could it be done?

Currently, the way to run TEX under any Emacs flavor is with the AUC-TEX package. The standard way for Emacs to talk to external processes is through the console. But if we aim at fast communication between TEX and the embedding environment, we might want to remove the overhead of accessing the console (on both sides). XEmacs is already able to dynamically load a DLL.

On the Win32 platform, one is expected to build shared parts of programs as a DLL, much more than on Unix. The TEX engine, for example, is built as a DLL because there are no symbolic links under Win32. So the DLL is linked to various stubs (`tex.exe`, `latex.exe`, etc.) that just set the program name which will be used later to load the right format.

So if we want to go further in this direction, what we need is a TEX program independent of any console, that could be addressed by other programs which will dynamically open the `tex.dll` and communicate with it through its API. At the moment it is not really easy because TEX does not know how to report errors by any other way than writing to the log file or to the console.

The big picture would be to make XEmacs able to display DVI files into one of its buffers, which might actually not be so difficult. This could be done by turning some previewer into a DLL and linking XEmacs to it. Then what you might want is the ability for XEmacs to talk to the TEX DLL, and let the output of the TEX DLL feed the previewer. Jonathan Fine will present his *Instant Preview* system in these proceedings, which reuses the ability of Web2C engines to send their output unbuffered through a socket. Jonathan's system splits the dvi output into smaller parts to achieve a better effect of instantaneously viewing the output of the text being typed in. If we go the way of sharing code in libraries and if some powerful DVI library is designed, then we could even remove the overhead of creating all those files, because the DVI library could allow any of its clients to access any page of any file currently known. A lock mechanism together with a copy on write mechanism could achieve even much better performance, and we would have a really nice typesetting system.

This is still fiction, but this kind of project can certainly be put up. So if you have students or man power to write code, please stand-up!

5. Conclusion

TEX used to be difficult to set up. Over the years, thanks to the many contributions of the TUG community, things have become smoother. However, the status is not yet up to the current standards of plug and play software. To go further in this direction requires us to rethink some of the core code.

I hope that by opening up the discussion on these topics I will have woken up some good will and that a new wave of contributions will appear: we need volunteers to make things move.

◇ Fabrice Popineau
Fabrice.Popineau@supelec.fr