fpTEX: A teTEX-based Distribution for Windows

Fabrice Popineau
Supélec
2 rue E. Belin
F-57070 Metz
France
fabrice.popineau@supelec.fr
http://www.ese-metz.fr/~popineau

Abstract

This paper deals with the ins and outs of porting the widely used teTEX distribution to the Windows environment. The choices made and difficulties experienced are related, a brief description of this huge distribution is given and the future work is sketched out.

Motivation

Context. More and more people need to use some sort of Microsoft environment, perhaps because of office suite or the like, or because of management staff decisions. Some of the greatest pieces of software have been developed on UNIX (or other operating system) well before the general availability of Windows. Thus, software such as TEX should be available on Microsoft operating systems, natively ported, and compatible with implementations on other operating systems. TEX by itself is largely platform independent but achieving complete compatibility for an entire distribution is better.

The Web2C TEX distribution is one of the greatest TEX implementations and has support for portability. Moreover, Web2C is the base of the widely used teTEX distribution for UNIX. Now that UNIX and Windows can easily share files across the network, thanks to tools such as Samba, it is most desirable to have not a teTEX-*like* TEX distribution for Windows, but the *actual* teTEX distribution for Windows.

Free TEX for Windows. In the fall of 1997, when I began to port Web2C to Win32, one main TEX distribution was available to Windows users: emTEX. This is still a great TEX environment but it was designed for MS-DOS first and then for OS/2. So, when Windows became 32-bit-aware,[1] those MS-DOS applications could not benefit from the new 32-bit flat mode, or at least not optimally. The so-called *DOS-extenders* were not as smart[2] as they are today. Some of the nicest features of emTEX, such as its `dvipm` previewer, were not available to Windows users. Moreover, emTEX's author, Eberhard Mattes, never released his sources.

At the same time, MiKTEX began to mature. Christian Schenk, author of MiKTEX, has followed a different way. He has designed a completely new Win32-oriented TEX distribution. Looking at his work, I questioned the usefulness of porting Web2C to Win32, but there were some reasons to do so:

Compatibility. Having a Windows TEX distribution based on exactly the same files as the UNIX one means you can *share* resources. For example, you can not only share `texmf` trees across the network but also configuration and format files.

Portability. Most of the ongoing developments around TEX are done with UNIX Web2C (see pdfTEX or ε-TEX). Being able to share source files means less efforts to compile a new release. There is another consequence: on several occasions, it has proven to be useful to compile the source code on something really different from UNIX. Errors that do not show up on one platform may do so on another one.

Usability. Lots of people are familiar with teTEX under UNIX. Having the very same distribution under Windows is a plus.

The plan. The porting tasks can be divided as follows. Note, however, that the job was not formally planned at all since it had to be done using mostly spare time. So the project has followed a circular technique, with some issues only being resolved quite recently. Below is a very short description of the next sections.

[1] In fact, even if Windows 9x can run 32-bit mode applications, only the Windows NT incarnation of Windows is a true 32-bit environment; see section about the previewer

[2] For example, support for *long filenames* was not available at first.

Command-line programs. The first goal was to have a `tex.exe` running under the Win32 API (Application Programming Interface), the set of functions that implement the `kpathsea` library.

Compilation environment. teTEX uses a powerful tool called `autoconf`. This tool relies heavily on having a UNIX shell and lots of UNIX utilities such as `sed`, `grep`, `awk`, and so on. Clearly, this is not something easy to find and run under Windows.

Shell issues. Moreover, the source distribution uses some shell scripts at *run-time*. It is not wise to suppose that the end-user will have a UNIX shell on a Windows machine.

Operating system-specific. Some issues like finding a replacement for file links or naming files on the network have been solved recently. The question of using the registry is also mentioned.

Previewer. No TEX distribution would be complete without a DVI previewer. So the port of xdvi was contemplated.

Installation. This is the trickiest part. Binary distributions were not common under UNIX but they are under Windows, and the installation process is very different.

Configuration. The process of configuring Web2C is very simple because it consists mainly of editing text files or setting up environment variables. The teTEX distribution introduces a smart tool to administer the system, and this task can be rendered in a Windows-oriented way as well.

Future work. There are many points that can be enhanced and some will be done in the very near future.

However, if tasks such as editing a text file, setting environment variables or unpacking an archive are usual in the UNIX world, they are not usual anymore in the Windows, world where end-users expect automatic or point-and-click things to happen. So the installation and configuration parts are very specific to Windows.

The contents of the distribution

Before discussing the porting issues, here is a brief outline of what is in the distribution:

- Web2C base distribution: TEX, METAFONT, METAPOST, DVIware and fontware tools
- each TEX extension or package that is found in teTEX:
  - ε-TEX, pdfTEX, Ω (Omega)
  - `dvipsk` and `dviljk` to print DVI files
  - `gsftopk` and `ps2pk` to rasterize Type 1 fonts to PK files
  - `mktex*` support programs for generating missing font files and `fmtutil` for building formats
- a DVI file viewer based on xdvi, but adapted to Windows
- packages found on the TEX-Live CD such as:
  - `dvipdfm`, to convert DVI files to PDF
  - `tex4ht` and `tth`, to convert TEX files to HTML
  - extra tools to deal with either DVI files, PostScript files or fonts.
- extra packages found only on the Win32 section of the TEX-Live CD:
  - `ttf2pk` and `ttfdump` will handle TTF fonts
  - `hbf2gf` will handle East-Asian fonts
  - `gzip` and `jpeg2ps`, which can be handy
- the teTEX `texmf` tree, which is not the least important part!

Command-line programs

The process. The Web2C distribution integrates all TEX-related tools around one main library called `kpathsea`. It was devised by Karl Berry to face the growing number of environment variables needed to set up a complete TEX distribution. Instead of setting environment variables, the path values and many other constants are looked for in a configuration file. This guarantees extensibility and is far easier to maintain.

The second point is the process of compiling TEX itself. The Web2C distribution owes its name from the Web → C translator that converts the original Web code to C programs. Basically, the following tools were needed:

- a C compiler targetting the Win32 API and, if possible, supporting the standard C library functions;
- some UNIX tools such as `sed`, `grep` and `awk`;
- the Perl language, which has proven to be useful to put glue between many parts of the building process, due to the lack of a shell with real programming capabilities under Windows.

Choosing a compiler. The availability of GCC — or rather of a native Win32 GCC — under Windows is quite recent. Moreover, GCC in its Cygwin[3] incarnation has some drawbacks under Windows:

[3] Most of the GNU tools have been ported to Win32 by *Cygnus Software* and their port is under the GNU Public Licence. See `http://sourceware.cygnus.com/cygwin/`.

- Every program is linked to a DLL (Dynamically Linked Library) that emulates UNIX calls; this slows down somewhat programs doing intensive file system calls,[4] for example.
- At the time I began the Web2C port, this DLL was not stable at all.
- Benchmarks on the same computer using GCC under Linux and the Microsoft compiler under NT have shown up to 20% less time on the same runs in favor of the Microsoft compiler.[5]

Thus, the Microsoft compiler was chosen. The general philosophy was to stick to the Win32 API as much as possible and avoid any layer to handle the translation, which might alter performance.

Many of the auxiliary tools needed for the build process were available either through the GNU-Win32 Cygnus project or from previous ports to MS-DOS; however, almost none of them were available natively ported to Win32. While I was at porting `kpathsea` and Web2C to Win32, I also adapted the tools I needed to Win32. This resulted in an archive of UNIX tools, many compiled by myself and the others gathered from the net. This archive is available in the same directory as fpTEX, in the CTAN archives.

Compiling `tex.exe`. The `kpathsea` library already had support at the source level for other platforms than UNIX, namely Amiga and VMS. So the path was already laid out. Fortunately, `kpathsea` already encapsulated almost all system calls needed for TEX. This was a great feature of the TEX source code, to precisely identify system dependencies.

Disks. The Windows environment knows about *device names* attached to disks whereas the directory tree structure under UNIX hides them.

Paths. The path separator is not the same but, fortunately, the Win32 API support '\' and '/' path separators.

Links. There are no hard or symbolic links under Windows.

Permissions. The permissions on files for UNIX and Windows are handled in a completely different way.

Some of the problems met were specific to Windows 9x, where standard C library calls are available but buggy. For example:

- `system()` is meant to run external commands but fails to return their exit code — it always returns true.
- `popen()` is available only for command-line programs but fails for graphical programs (the previewer uses this call!).
- `stat()` fails to recognize directories if their name has a trailing `/`.

All these problems have workarounds using the Win32 calls instead of the standard C calls.

Compilation environment

In order to make the build process safer and closer to what happens under UNIX, a number of decisions had to be made.

Makefiles. Every UNIX `Makefile` comes in a generic shape, `Makefile.in`, that needs to be instantiated by the complex process of `autoconf`. The UNIX `Makefile.in` is assembled and processed by the `m4` macro processor to generate the actual `Makefile` that will fit your own configuration. Moreover, those `Makefile` use many UNIX constructs (shell or other tools). So they are not usable as-is under Windows, where the process is somewhat different.

The Windows `Makefile` is built by hand from the UNIX `Makefile.in`. All common parts are stored in a special place. An initial `configure.pl` Perl script allows one to:

- configure the common parts with options like root of the destination directory, root of the source tree, and so on;
- ensure through the configuration that only the files generated by the current build will be referred to; that is, no external `kpathsea.dll` will interfere, no external `pdftex.exe` or `texmf` will be referred to when generating documentation or file formats;
- save and restore each of the `Makefile` files in a safe place.

Source code configuration. The same Perl script also undertakes the translation process of every `config.in` or `c-auto.in` configuration file into their definitive form. Since there is only one target operating system, there is no need to guess if the features are supported — just consult a table of features. Thus, doing this ensures better compatibility with the original source code.

Overall build process. The overall build process is done by another `build.pl` Perl script. This script delegates to the different `Makefile` and can be asked to:

- clean up the source tree at different levels
- rebuild dependencies
- build and/or install the whole release

[4] The `kpathsea` library can do lots of `stat` calls on a huge `texmf` tree.

[5] This was GCC 2.7.1 versus VC++ 5.0; the situation may have changed today.

- use different compilation modes, such as *debug* or *release*, *statically* or *dynamically* linked executables
- prepare for specific tasks, such as *profiling* or using advanced debugging and checking tools such as BoundsChecker
- install everything *from scratch*, including installing the latest teTEX `texmf` tree

Up to now, the build process is not clean enough, but nonetheless the source tree has been used successfully by people with no previous knowledge. Cleaning up the Win32 part of the source tree would allow more people to access it and contributions from the net could be expected.

Shell issues

The Web2C distribution may ask for font generation at run-time. This is done through the `kpathsea` library calling an external command when it fails to find some needed font.

Because of the complex and evolving nature of this process — how many versions of those `mktexpk` scripts have been devised? — the generation of fonts has traditionally been handled by shell scripts.

The shell requirement needed to be removed under Win32 and the Perl alternative, despite some drawbacks, was considered:

- Perl provides greater portability across such different operating systems as UNIX and Windows;
- Perl is not widespread enough under Windows and under UNIX, which means you can easily find it but not every single user will have it or want it;
- Perl has quite a large disk space footprint.

Had UNIX users been ready to switch from shell scripts to Perl scripts for this task, things might have been different but that was not the case. So the only risk-free and simple solution from the user point of view was to code the shell scripts in C. Given the complexity, the C version of the scripts required several rewritings before becoming reliable enough. But now, they do behave like their original shell counterparts. And the C version of these scripts can even be used under UNIX. The several `mktex*` shell scripts are provided as one DLL, with several stubs,[6] following the same philosophy as for the TEX engines — see next section. This means that `kpathsea` could be linked with this `mktex.dll` and could avoid creating a new process to generate a new font file.

[6] A *stub* is a small executable program linked to some DLL and whose only function is to set up some parameters before calling the DLL. This way, the same large DLL can be called in different ways by using only small executable programs.

Operating system-specific

Two main features have been added and one has been avoided.

No file links under Windows. The Web2C distribution uses file links under UNIX for linking programs under different names. The problem is that you can have several format files generated by one engine. For example, `latex.fmt` and `plain.fmt` are both run with the `tex.exe` engine. Under UNIX, the `tex` engine maybe linked under the names `latex` and `plain`, and the name under which the engine is linked determines which file format is loaded.

There are no file links under Windows [7] and all you can do is simply copying the `tex.exe` engine to `latex.exe` and so on — at the expense of disk space. Given the number of engines, some of them being quite large, it is important to overcome the problem of file links.

Fortunately, for executable programs, there is a natural way of doing something similar to file links using the Win32. The trick is to build a DLL with all the engine code and to have a small stub linked to the DLL. This way, the DLL is shared and the stub can be copied without using too much disk space. For example:

```
03/17/99  08:44a    16,384 pdfinitex.exe
03/17/99  08:44a    16,384 pdflatex.exe
03/17/99  08:44a   389,120 pdftex.dll
03/17/99  08:44a    16,384 pdftex.exe
03/17/99  08:44a    16,384 pdfvirtex.exe
```

There are four stubs linked to the same DLL. Should you create a new format file called `frpdflatex.fmt`, for example, you only need to copy `pdftex.exe` to `frpdflatex.exe` and the new format file will be loaded automatically by calling the new command, which has a very small footprint on disk.

There are other potential advantages:

1. upgrading to a new version of pdfTEX could be done by only upgrading `pdftex.dll`;
2. clever TEX shells could drive TEX engines directly by talking to the DLL and not use the command line.

Accessing files on the network. Under Windows, you can access files shared on the network by using UNC names. UNC refers to *Universal Naming Code*, a syntax introduced by Microsoft to refer to shared resources — files or devices — available through the network. The `kpathsea` library is thus made aware of UNC names, means you can make

[7] Shortcuts are not file links but rather redirections, available only through the Windows shell environment, not from the command line.

`$TEXMF` point to `\\TeXServer\TeXmf` or ask `dvips` to print on `\\TeXServer\printer`.

The registry. Under Windows, every program accesses the registry to retrieve its parameters and all required information. The registry is a database, shared across the network which encompasses the environment.

So the question arises: should the port of Web2C to Windows use the registry? The answer is no. The main reason is that it is not recommended that end-users modify the registry by hand — there is too much potential danger for their system. So storing the configuration into the registry would prevent users to easily change the way their tools behave. Temporarily setting environment variables is a quick way to modify `kpathsea` behavior and a very useful feature which it would have been unwise to remove. Users can fiddle with their configuration parameters exactly in the same way they would under UNIX: by either editing the `texmf.cnf` file or overriding parameters in the environment. It is always a matter of getting the best of both worlds.

Previewer

Motivation. It was not at first my intention to devise one. DVI format is not a *modern* format anymore. Even if it fulfills everybody's needs, it does not mean it will last. The new pdfTEX extension has demonstrated that DVI is not mandatory for a TEX system. Thus, devising a previewer for Win32 is not a simple task. Spending lots of time on a tool that might turn quickly into something obsolete is not very appealing.

But no TEX distribution would be complete without a DVI previewer: many TEX users stick to the good old (plain- or LATEX-generated) DVI format before any kind of PostScript conversion.

We still might argue that Ghostscript provides an accurate view of what will be printed, but the process of TEX → `dvips` → Ghostscript is somewhat slow and heavy for many documents.

So I ended up in looking at the xdvi source code. As I had no previous experience of Win32 *graphics* programming nor of X-Window *graphics* programming, so this was another reason for doing the previewer.

Porting *graphics* application. If we omit the interface, xdvi uses only a few primitives from X-Window: it only needs to draw bitmaps for glyphs and rectangles for rules. So the decision was made to adapt to Win32 everything that could be (the page reading and drawing mechanism, for example) and to rewrite the user interface part.

All but two of the C source files have been patched to compile under Win32 and the missing graphic primitives have been added. As well, a new user interface has been devised following the samples provided by Microsoft with their Win32 System Development Kit.

Some issues have been raised — and solved! — by the redisplay mechanism. The main problem with the redisplay was where it should happen: in memory or directly onto the display surface? The former was easier but had one major drawback: at a scale factor of 1 and 600dpi, an A4 color page would be huge (about 34Mb). So, on a second try, the redisplay was changed to draw directly onto the screen. In fact, both solutions are still in the source code but only one is activated.

This is also the same reason for Windvi not displaying PostScript inclusions at a scale factor of 1: Ghostscript is told to allocate the whole page because it can be asked to display raw PostScript code. So this time, Ghostscript would require the 34Mb page. It does work under Windows NT — albeit very slowly — but it is too heavy for Windows 9x.

This was also the opportunity to fully understand where Windows 9x and Windows NT are different. They share the same API but they behave in very different ways. For example, the first data structures I built and that used to work under Windows NT assigned one bitmap handle per glyph used in the DVI file. It was even pretty fast but the same program running under Windows 9x was slow and eventually crashed. Looking at the resources, all the graphics resources were used. How to explain such different behavior? In fact, the Windows 9x GDI — the kernel part that implements graphics services — allocates all the graphic objects in a few 64K stacks. The one dedicated to bitmap headers was quickly filled in when the DVI file was using even a low number of fonts.

Features. The features of Windvi are almost the same as those of xdvi. I have tried to mimic the behavior of xdvi whenever possible and, at the same time, to add Windows behavior via *status bars*, *tool bars* and *tooltips*. The most important features of Windvi include:

- monochrome or grey-scale bitmaps (anti-aliasing) for fonts
- easy navigation through the DVI file
  - page by page
  - with different increments (by 5 or 10 pages at a time)
  - go to home, end, or any page within the document

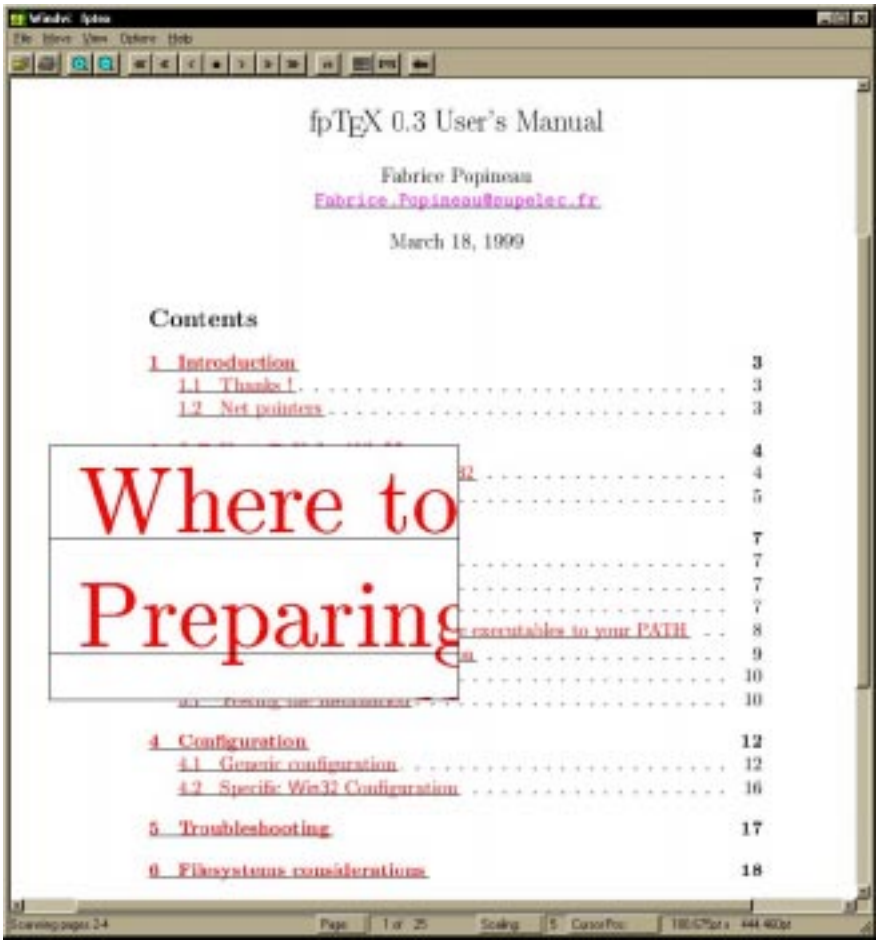

Figure 1: Windvi featuring magnifying glass and Hyper-TEX links

- different shrink factors to zoom page in and out
- magnifying glass to show the page at the pixel level
- compatible with xdvi keystrokes
- use of `.vf` fonts
- display `.pk` and `.gf` font files
- automatic generation of missing `.pk` files, even for Type 1 fonts
- tracking DVI file changes and automatic reopening
- understanding of Ω extended DVI files
- drag-and-drop file from the Windows shell explorer
- external commands through `\specials`
- color support (*à la* `dvips`)
- real-time logging of background font generation
- visualization of PostScript inclusions
- support of Hyper-TEX specials
- printing.

The main features not found in xdvi are color support and printing. The latter was again the opportunity to test different behaviors between Windows 9x and Windows NT — quite painful to debug. In fact, printing is something not at all easily done because there are many ways to handle it:

1. Print through the generic Windows printer driver, but PostScript specials will not be printed.
2. Ask `dvips` to convert the `.dvi` file to PostScript, and then either send the file directly to the printer, if it can handle it, or else call Ghostscript to do the job.
3. Build a bitmap with the page, PostScript specials included, and do *banding* (because the page would be huge) and send the bitmap to the printer.

Currently the first and third options are implemented but the third one uses lots of Windows 9x resources.

Last, the fact that Windvi is quite close to a port of xdvi was rewarding when it came to implementing the Hyper-TEX feature, which relies on the use of the `libwww` library, maintained by the W3C consortium. Fortunately, this library is available for UNIX *and* Windows. The net result was that adding this Hyper-TEX feature to Windvi took only a couple of hours to have a first workable result that allows navigation inside the document and referencing external programs. However, it turned out that, under Windows, this library is not mandatory at all because the shell can be called to open URLs, so it is not needed anymore.

Installation

Packaging TEX. Maintaining a `texmf` tree is a job that is very well done by Thomas Esser for teTEX and by Sebastian Rahtz for the TEX-Live CD. Given such a tree, I wanted to find a way to *automatically* group files by packages.

As has been pointed out in electronic discussion lists,[8] there is a lack of a standard procedure to install TEX packages. So there is no way to get a *source* `texmf` tree and *build* it, logging where every single file has been installed. So I wrote a few Perl scripts to reverse-engineer the build process, keeping the following goals in mind:

- Have three levels of completion: *basic*, *recommended* and *full*; given a package, these levels are guessed by heuristics from the `lists` files, devised by Sebastian Rahtz and to be found on the TEX-Live CD.
- Build a two-level structure targetted for InstallShield us (see next section); this structure is based on *components* (e.g. `latex`, `omega`, ...) and *subcomponents* (e.g. `latex\graphics`).
- Group files as much as possible; for example, to group fonts files, style files, source and documentation files for one package. This was done by implementing some kind of rule-based system in Perl, along with some other ad-hoc rules.
- Give a description for each sub-component; this was done using the Web description of TEX packages assembled by Graham Williams.

The result is not perfect, especially for the TEX-Live CD, with its huge number of packages. Notably, the automatic detection of descriptions is flaky — some of them being false, but this is harmless — and the recommended installation installs far too many packages, which means that the levels attributed to packages have been underestimated.

[8] See the dedicated mailing list on `tug.org`.

The installer. There is a product dedicated to building installers for Windows that is widely known and used in the Windows world, called InstallShield. Given a tree structure of *components* to which are associated *file groups* and a few *setup schemes*, it will build a nice looking installer.

But things are not so easy when it comes to installing a huge distribution. While InstallShield handles many common installation cases well, TEX is quite special because of the large number of files. This provides the opportunity to experiment with bugs and any limitations in InstallShield. Even though it is being used for the current release of fpTEX and the TEX-Live CD, it will probably be abandoned and replaced by a dedicated installer.

All in all, even if the installer is not perfect or flexible enough, it is useful enough to install from the TEX-Live 4 CD. The latest version of the installer used on fpTEX even makes it possible to add packages on top of an existing installation. And finally, it has been useful to use InstallShield to sort out all the problems related to installation; even if it is not used for future versions, the specifications are still there.

Windows integration. Most environments dedicated to TEX in one way or another will support fpTEX. Amongst them, we can cite WinEdt, 4TEX and XEmacs.

Configuration

Assuming a *recommended* installation, there is little to configure. But, as pointed out in the introduction, Windows users expect dialog boxes not text files to be edited. This has lead me to devise a dialog box-based tool targetted at fpTEX configuration. The `texconfig.exe` tool allows the user to access most of the configuration files in a point-and-click way.

Moreover, the standard Windows menus are provided with shortcuts to command-line tools (to rebuild formats or file database) and to local web pages (to access the documentation).

Future work

Some of the above-mentioned components will be enhanced in the near future.

Previewer. Even if the DVI format is old nowadays, many people are still using it so I will enhance Windvi in the following ways:

- Type 1 and TTF font support
- other graphics files format support
- graphical transformations for glyphs and rules under Windows NT

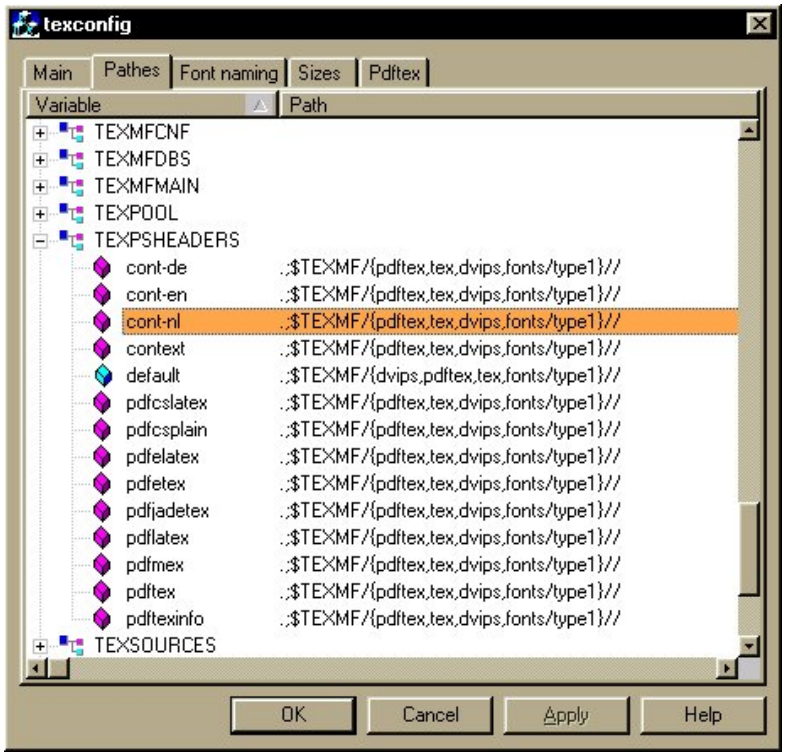

Figure 2: `texconfig.exe` tool editing pdfTEX related paths

- two-page spread mode
- forward and inverse search; that is, from the editor to the DVI file or from the DVI file to the editor
- cut-and-paste to other applications

Installation. The dedicated installer is being worked on. It will handle installations of both fpTEX and TEX-Live. The fpTEX files will be distributed as `.zip` files and if they are not present, the installer will try to download them from the net.

Configuration. The `texconfig.exe` program is not yet available. Its interface needs to be discussed because it is difficult to be simple and powerful at the same time. Many users want to tweak the configuration whereas, to make this thing really simple, we should hide most of the configuration parameters.

Availability

The whole package is available from CTAN, in the `systems/win32/fptex` directory. More information is available from `www.ese-metz.fr/~popineau/fptex`, the home of fpTEX. The TEX Users Group is kindly hosting a dedicated mailing-list, `fptex@tug.org`, to which you can subscribe by sending a request to `majordomo@tug.org`.

Acknowledgements

All this work relies heavily on the work done by Karl Berry, Thomas Esser, Sebastian Rahtz and Olaf Weber on the UNIX distributions Web2C, teTEX and TEX-Live.

Obviously, the numerous authors of all the packages present in fpTEX — programs or TEX style files — are thanked too for having shared their work.

References

Berry, Karl and O. Weber. “The Web2C distribution of TEX”. `http://tug.org/web2c`, 1999.

Esser, Thomas. “The teTEX distribution of TEX”. `http://tug.org/pub/teTeX`, 1999.

Gurari, Eitan M. “A demonstration of TEX4ht”. `http://www.cis.ohio-state.edu/~gurari/tug97/tug97-h.html`, 1997.

Popineau, F. “Rapidité et souplesse avec le moteur Web2c-7”. *Cahiers GUTenberg* **26**, 96–108, 1997.

Popineau, F. “Windvi User’s Manual”. *MAPS* **20**, 146 – 149, 1998.

van Dobbelsteen, G. “DVIview: A new previewer”. *MAPS* **20**, 120–124, 1998.