# TEX Live under Windows: what's new with the 7th edition?

Fabrice Popineau
SUPELEC
2, rue E. Belin
57070 Metz
FRANCE
fabrice.popineau@supelec.fr

**Abstract**

The 7th edition of TEX Live under Windows has some new features that are explained in this paper. Especially notable are two experiments to extend Kpathsea beyond its original abilities:

- sharing Kpathsea data-structures between several processes for faster processing,
- allow url's in client programs as well as filenames whenever they ask Kpathsea to open a file.

Additional points about the `TeXSetup.exe` program and the evolution of other parts of the distribution will also be discussed.

## Extensions to Kpathsea

**Rationale** Kpathsea is at the heart of any Web2C based TEX distribution. The idea of gathering all the services needed by the TEX family programs around a common API was a great step forward, compared to the previous TEX distributions. Kpathsea features have been quite stable since version 3: there have been only minor changes. The syntax of the `texmf.cnf` file is the same, the file searching algorithm has converged to the current one, which may not be entirely satisfactory, but which is at least stable.

One of the Web2C features – the `\write18` command – triggered new uses of the whole TEX system. Specifically, it became possible to make `tex` call `metapost`, which will on its turn call `tex` again to typeset labels. As one document can hold many hundreds of metapost figures, we may wonder about the overhead when starting Kpathsea. In fact, we already noticed that the Kpathsea initialization time is far from insignificant. If we take the full TEX Live installation as a reference, then we have quite a long `texmf.cnf` file to parse and a huge `ls-R` hash-table to build, which are responsible for most of this initialization time. So for a given job, which needs to run `tex` and `metapost` in sequence for hundreds of figures, we can wonder if we could cut down the processing time by avoiding to repeat this initialization sequence: most of the data structures that Kpathsea will build across a single job will be the same for each instance of the programs.

A second extant issue with Kpathsea is how far the path notion extends. From the beginning, Kpathsea was written to handle any kind of paths for any kind of operating systems: IBM MVS, DEC VMS, Unix, Amiga, DOS, OS/2, Win32. Not all of them are still actively supported, but they could be so in principle. Among them, Unix has probably the simplest, most regular path syntax.

But supporting Win32 means that UNC share names like `//Server/SharedDirectory/` have to be supported, as well as `c:/Program Files/TeXLive/texmf/tex/latex/base/article.cls`. Looking at these pathname examples, we can see that each of them can be divided into 3 parts: a source (server, shared name, drive), a path relative to the source and the file name. So given that the URL syntax also follows the same pattern, we can wonder why it could not be supported by Kpathsea.

A bare implementation of both features is active in the Win32 version of Kpathsea which is available on TEX Live 7. The description of their implementation and related problems follows. It is important to keep in mind that all the changes with report to the standard TEX Live sources have been made with simplicity in mind. Sometimes better solutions could have been found but at the price of more extensive changes or code rewriting. All of the TEX Live specific Win32 changes are provided by the `/source/source-win32-patch.tar.bz2` file on the cdrom.

**Extending the "path" notion** What do we want to achieve in extending Kpathsea towards the Internet? The basic usage could be something like this:

```
\includegraphics{http://server.net/image.jpg}
```

It could also be interesting for files included in a document to be taken from the Internet. We can wonder how long such a url might last (the Internet is moving so fast), but in the meantime more and more people are processing XML documents using TEX and so it can be useful to retrieve XML fragments off the Internet. Given this, we need to cope with only 3 kinds of url's: `file://`, `ftp://`, `http://`. The others are not useful. The first one is not a problem because it is only syntactic sugar for local files.

So we want at least that any file opened for reading by TEX can be replaced by an `ftp://` or `http://` url. What does this imply from the file point of view? Playing with remote files introduces unknown behaviour because the files may not be available. Moreover, both ftp and http protocols have different features. The http protocol allows for retrieving file information (availability, size) without transferring the actual file, whereas the ftp protocol cannot feasibly do this.[1]

The simplest solution for implementing remote file access is to use a third party library that downloads the file locally. Downloading the file at opening time will ensure future availability. All the remote files are given temporary names and the association between url's and temporary names is stored in a hash-table. When Kpathsea exits, all such temporary files are removed. This is far from optimal, but safe enough for testing the feature. We could enhance the process by managing a cache of downloaded files, avoiding download of files that are already present unless they are too old and so on, but this is probably not a job for Kpathsea: if such a feature has to be used intensively, then it would be better to have a high-performance cache system for your whole Internet connection.

The question is now the following one: where to hook in the function that will retrieve remote files? Given that we essentially expect TEX and friends to be the programs using remote files, we could hook into the `web2c/lib/openclose.c` file which holds the input/output functions for TEX programs. However there are also reasons to hook inside Kpathsea:

- we need to extend path parsing and this is Kpathsea job;
- we can wonder about the opportunity of making Kpathsea Internet aware and be able to set up a remote texmf tree for example;
- other programs than TEX engines could benefit from this feature;
- under Win32, it is easier to replace `kpathsea.dll` if we want to enhance it or fix it.

So we chose to make the changes inside Kpathsea rather than inside the TEX engines. The internals of Kpathsea have already a few macros and functions used to parse various kinds of paths, so adding a couple of macros and a few cases inside these functions was easy. Given the fact we want to download remote files as soon as they are accessed, we trap the `fopen()` call and hook file downloading there. This way, any Kpathsea file can be replaced by a url. This is enough to provide absolute remote file retrieval. However it is not enough yet to provide remote file searching. This would require to trap also the `opendir()` and `readdir()` calls. Although internet aware versions of these calls are provided by the Win32 WinInet API, this change has not been implemented yet[2].

The only remaining problem was to find a reliable library to download files. There are several choices:

**libwww** the W3 Internet protocol library, which is Unix and Win32 compatible. After quick testing under Win32, file downloading was unreliable and blocked sometimes. Moreover, even after careful documentation reading, we have never found out how to display download progress, although it is said to be possible;

**libcurl** this library is provided together with a command line tool and seemed to be very handy. It has been confirmed at least under Win32 where it worked like a charm at first try;

**wininet** the Microsoft Internet library provided with Internet Explorer 5 and later. This is solid and has a high-level API, but Win32 only;

**sockets** raw protocol handling is possible too but if we want reliability, better to rely on some higher level library.

Ideally, it should be possible to plug any new library into the system and choose the one used from `texmf.cnf`. We need only one function to ensure the compatibility layer:

```
int get_url_to_file (
  /* the url of the file to download */
```

[1] Retrieving the file size requires parsing a directory listing, but there is no standard for this listing.

[2] In fact, because of the overhead introduced by initializing `ls-R` hash-tables for a remote texmf tree, this could be interesting in the larger framework of a Kpathsea server. See Karel Skoupy's talk on this topic in this same proceedings.

```
char *_url,
/* the temporary filename */
char *_filename,
/* if we know the file size */
int expected_size,
/* the function to log the download */
pfnLog log,
/* the function
   to report download progress */
pfnProgress progress,
/* the download method to use */
int method
                  );
```

According to the method selected one or another library will be used to download the file. Currently, wininet is hardwired as the selected method but work is under progress to provide the same feature with libcurl under Unix and as an alternative to wininet under Win32.

The feature works quite well and is easy to use. Enhancing it will require a careful analysis of its uses. For example, it is common usage to open a file to test its availability. In this context, the file will be downloaded unconditionally, which we might want to avoid.

**Kpathsea and reentrance** The initialization time problem would have been easily solved if Kpathsea had been reentrant, but this is far from being the case. Kpathsea has been designed for quite a few years now and that feature was certainly not given a high priority if envisaged at all. Making Kpathsea fully reentrant is certainly not an easy task without major rewriting: that would mean for example that different programs – say `latex` and `context` – could call it simultaneously to look for files and it would be able to answer both of them. For the moment, unfortunately, the calling program, upon which search paths depend, is defined at initialization time, and the current data structures make it difficult to redefine it afterwards.[3]

Another point with initialization time is that it is even longer under Win32 because the file systems (FAT32, NTFS) are case-insensitive (though they preserve case). Hence, filenames read from the `ls-R` files are stored in the hash-table after conversion to uppercase. But since Win32 is able to handle Unicode and MBCS, the conversion realized by the function `toupper()` is as slow as you might expect (it has been measured several times slower than the same function provided by the GNU C library which did not handle MBCS or Unicode).

[3] Actually, it can be done and the C version of the `fmtutil.exe` program does it, but it can work only in limited cases.

Solving our startup time problem is, however, quite simple. All we need is to prevent Kpathsea from rebuilding all of its hash-tables across a single run of several Kpathsea-linked programs. Here is the list of these hash-tables:

1. configuration parameters, essentially every variable/value pair read from the `texmf.cnf` file;
2. `ls-R` database, the huge (around 40000 for TEX Live) list of filenames and paths;
3. alias database, this is a list of alias names for a few files;
4. fontmap database;
5. links database, this is internal data used to accelerate the recursive search on paths;
6. symbols database, this is the list of pairs of variables and values used to emulate the various `mktex...` shell scripts in C (Win32 specific);
7. remote files database, the remote files that have been downloaded from the Internet and stored locally under a temporary name (see previous section).

Storing all these hash-tables into a shared memory block allows us to avoid rebuilding them. If the block does not exist yet, it is then initialized. If it is there, then we can bypass Kpathsea initialization and use the already built hash-tables. Managing a shared memory block has some drawbacks however:

- the shared memory block is allocated once and for all, all you can do after it is built is release it; it is not extensible. This is because it needs to be allocated at the maximum size that will be needed, or else it is quite inefficient to resume from the situation (see below);
- all pointers must reference data relative to the beginning of the shared memory block: there is no way to know the address of this block for each of the processes using it;
- none of the standard malloc facilities (`malloc`, `free`, `realloc`) are provided to play with memory inside this shared memory block. Fortunately, managing the hash-tables only requires that we allocate memory. Although occasionally we might need to remove an item from a hash-table, we can safely ignore freeing this memory because the situation does not occur frequently.

Managing data inside the memory block required a few changes to Kpathsea functions. All data are reallocated inside the shared memory block, so there is no point in storing a pointer to preallocated memory as was the case in a few places. When using the hash-tables, all values returned are

`const char *`: it is the user's responsibility to duplicate them to become writable. All of this has been done by introducing a new module replacing the `kpathsea/hash.c` one, with a slightly different, more regular API for the hash-tables functions. Then, all the calling functions were adapted to this new API. Managing relative pointers inside the shared memory block has been done by macros, but all pointers returned by the hash-tables function are absolute addresses inside the virtual process space.

Shared memory blocks are not easily extendable. Given that our block is identified by its name, extending it requires allocating a second one with a new name, copying data from the old one to the second one, releasing the old one, creating a third one with the old name but larger than the first one, and finally copying the data from the second one to the third for the second time! Moreover, allocating a new block supposes we can propagate the new base address which can become tricky. In fact, we are only able to extend the shared memory block at initialization time (so no propagation needed) using the previous mechanism. Shared memory block extension will occur only if the block is filled when reading the hash-tables, or if there is not enough free space after initialization is done.

The `mktexupd` program has been enhanced to add directly the new file inside the already loaded hash-tables because it is merely an addition to the existing list of files. But to cleanly allow to rebuild all data structures for `mktexlsr` while the shared block is loaded would require careful redesign of Kpathsea (especially to be able to reinitialize it at any point).

One side effect of using this feature has both a positive and a negative aspect. If you have some program like the `windvi` viewer which is using Kpathsea, then the Kpathsea dll will not be unloaded until the viewer is closed. As long as an instance of the dll is running, the shared memory block will remain. So the shared memory block will be available for all TEX (and other Kpathsea-linked programs) runs that will occur and you will gain initialization time at every run. This is the positive aspect. The negative aspect is that you need to close all programs linked to Kpathsea whenever you want to change your configuration in the `texmf.cnf` file or whenever you need to run `mktexlsr`. Again, the hash-tables are built once and for all, and given the lack of real memory allocation operations inside this shared block, we cannot rebuild them easily unless *we shut down all the programs using Kpathsea.*[4]

The fact that `ls-R` files are not read again when the Kpathsea dll is loaded is minor inconvenience. It could be part of the managing environment to ensure that all processes are shutdown when `mktexlsr` is run. The `kpsecheck` tool is provided and reports about the shared memory block usage (see the section Other programs below).

### Windows-specific TEX Live features

**Small enhancements to Web2C** MiKTEX has had for some time options to the TEX family of programs that have not been available under Web2C. In order to enhance compatibility between both distributions, we added these options to the Win32 Web2C programs too.

The most noticeable one is the `-job-name` option. Using the standard Web2C TEX program (version 7.3.7), there is some inconsistency with the `-fmt` option:

- in virtex (or normal) mode, this option tells TEX to load the format specified,
- in initex mode, it requires TEX to dump the format under the specified name.

But we could also require to load some format and specify the dump name. In fact, the `-job-name` option solves this problem, because it allows to change the internal `\jobname` value, which is used both in ini and vir modes. So under Win32, the `-fmt` option has only one meaning: preload the specified format and the `-job-name` option can be used to change the base name of the files output by TEX. All of this is whether you are in ini mode or not.

The other options introduced under Win32 are:

**-halt-on-error** stop at the first error;

**-job-time=FILENAME** set the job time by taking FILENAME's timestamp as the reference;

**-output-directory=DIR** use DIR as the directory to write files to;

**-time-statistics** print processing time statistics about the current job.

**The `TeXSetup.exe` program** `TeXSetup.exe` has been made more reliable, especially for installation under Win2K/XP. Installation can be done for all users, but then you need to be a Power User or an Administrator, or install it for single users. The default location is `c:\Program Files\TeXLive` to

[4] Apart from `windvi`, there is another resident program which is linked to Kpathsea: it is `ispell`, the spell checker. If you run it from Emacs, you are bound to forget that `ispell` is running in the background.

be Windows compliant, but installing there can be done only by Power Users or better. If you have only standard rights, you can't install under the `c:\Program Files` default location but you can install in `c:\TeXLive` for example.

Win2K/XP rights behaviour may not be very intuitive, especially if you come from the Unix world. Any object rights are inherited from its container object at creation time, except if specified otherwise. The problem with TEX and especially when `mktexpk` build font files, is that when an object is moved, this object keeps its original rights. So if we apply this rule to `mktexpk`, the `.pk` is built by user A in the user temporary directory, so inherits user A's rights, then it is moved in the `$VARTEXMF` directory, hoping it will be available for everybody. But the file has kept user A's rights, so another user has no rights to access it. Worse, another user can't even overwrite it: the file is there and is neither accessible neither removable. To overcome this problem, we came up with this solution:

- at setup time, `TeXSetup.exe` creates the temporary directory `c:\Program Files\TeXLive\temp` directory and assigns it full access rights for everybody;
- the `TEXMFTEMP` variable is assigned the value of the temporary directory;
- at initialization time, Kpathsea substitutes the value of `TEXMFTEMP` for the standard `TEMP` value.

This way, all temporary files are created in the `TEXMFTEMP` directory, so they are given full access rights to everybody. Font files will keep these full access rights when moved to the `VARTEXMF` directory. That means everybody can remove them too, but this is not a problem since these files will be rebuilt at runtime when they are not available.

TEX Live 7 introduces so-called schemes of installation. Previously, we had collections and packages. In order to make things easier, we can now require that some predefined set of collections and packages be installed. Such a set is called a scheme. We have defined a couple of them: texlive-basic, texlive-recommended, texlive-full for standard texlive distributions of different sizes, but there are also the GUST scheme, the GUTenberg scheme and the XML scheme. What is interesting is that you can run batch installation using this kind of command:

```
c:\>TeXSetup --scheme=texlive-full --quick
```

It is easy to add new schemes by creating a new TPM file in the `texmf/tpm/schemes` directory. Complete documentation for the `TeXSetup.exe` program is provided with the TEX Live manual.

**Other programs** TEX Live holds quite many Perl scripts which are widely used: `texexec`, `texutil`, the new `updmap` (rewritten in Perl for Win32 from the original shell script version), etc. Up to now, you had to have Perl installled to be able to use them. Unfortunately, this is not an ideal solution, since:

- Perl is not a standard part of Windows;
- Perl is not available on the TEX Live CD-ROM because of disk space, although it is available for installation from the Internet as part of the Win32 support packages;
- if Perl is not installed, the scripts are not usable off the CD-ROM.

So with TEX Live 7, we decided to compile all the Perl scripts into `.exe` files using the ActiveState Perl compiler. There are drawbacks in doing this:

- the size of `.exe` files is quite large, even if the `perl56.dll` needed to run them is provided in a common `bin/win32` TEX Live directory and not included in the `.exe` compiled file,
- users cannot easily upgrade the Perl scripts, because they would need to be recompiled.

However, this is currently the only way to run those scripts off the CD-ROM on a machine where Perl is not installed.

Another new point with TEX Live 7 is the ability to build format files at run time. This was quite easy to setup inside Kpathsea because the mechanism is the same as the one used for fonts: when the requested file is missing, the specified command is run to try to build it. So in this case, we only needed to add the new `mktexfmt` command which actually behaves like the `fmtutil` one. The only difference is in the calling sequence:

```
c:\>fmtutil --byfmt=latex
```

versus

```
c:\>mktexfmt latex
```

and also the |mktexfmt| needs to write the path of the generated file to `stdout`. This `mktexfmt` command was easy to build out of the `fmtutil` shell script (Unix) and the `fmtutil.exe` C program (Win32). There are several benefits in doing this:

- no need to run `fmtutil -all` at the end of installation anymore (even if it is still done), since format files will be built on demand;
- fewer `.bat` files for all kinds of formats (those `.bat` files used to build the format file when it was not found),; for example, you only need to copy `tex.exe` to `mllatex.exe` and the command `mktexfmt mllatex` will be run if the `mllatex.fmt` format file is not found, provided your `fmtutil.cnf` file mentions it. Then, the

execution will continue. It even works for nested dependencies between format files.

Lastly, there is a new tool call `kpsecheck` which can report on a few aspects of TEX Live setup:

- the state of the Kpathsea shared memory block (in a future version this should include the processes using it);
- the Ghostscript location: Ghostscript is used in many places and it is handy to make those programs find it automatically;
- duplicates among the files stored in the hash-tables. Called with no argument it will report all the duplicates, but many files have the same name (`README`, etc.). So you have the option to include or exclude files based on globbing:

```
c:\>kpsecheck -multiple-occurences \
              -include=*.sty
```

to find all duplicates among `.sty` files.

### What is coming for the 8th edition?

It took a few years to stabilize the whole Win32 TEX Live. A few points are still missing, the most obvious involving the TPM files and the `windvi` previewer.

Talking about the previewer, the main missing features are:

- source specials support,
- Type1 and TTF font files support,
- safe printing and `dvips` printing.

Source specials support and Type1 font support are available from XDvi now. So importing this into Windvi will be quite easy.[5] TTF font files support will require a bit more work but it should follow the same global scheme as for Type1 fonts. Printing is an area that will require careful testing. The most annoying point is the huge spool files potentially created. This has been partially avoided by doing banding, at the cost of slowness.

[5] A first version of `windvi` supporting these features will be demonstrated at the conference.

The setup scheme could be enhanced in various ways. The granularity of TPM files is not optimal: binaries should be split into smaller packages. There are annoying dependencies between some packages (for instance, `aeguill` relies on files provided by other unrelated packages, like the Polish fonts). It is difficult to automate the construction of these packages from what is available on CTAN, even if Sebastian Rahtz has done a great work in this area. Probably we need a whole new infrastucture to submit packages to CTAN but that is another story.

Finally, starting next October, we will work on a project of tight integration between XEmacs and TEX in order to provide an easy-to-use out-of-the-box word processing tool. The goal of this project funded by the French Ministry for Education is to draw certain kind of people (mainly mathematics and physics teachers) towards TEX, on the premise that what refrains people to use TEX is not the TEX input syntax, but the complexity of installing and maintaining a TEX distribution. So we will do our best to provide a well documented, regular TEX system with integrated viewer. More to come about this project next year!

### References

Popineau, F. "fpTEX: a win32 port of teTEX". In *TUGboat*, volume 20. TEX Users Group, 1999. Proceedings of the 20th Annual Meeting of the TEX Users Group, Vancouver.

Popineau, F. "Directions for the TEXLive Software". In *Proceedings of the EuroTEX 2001 Conference, Kerkrade, the Netherlands*, pages 151 – 161. 2001.

Rahtz, Sebastian. "The TEX Live Guide, 7th edition". Available on the TEX Live 7 CD-ROM, 2002.

Skoupy, Karel. "TEX file server". In *Proceedings of the 23rd Annual Meeting and Conference of the TEX Users Group*. 2002.