MMTEX: Creating a minimal and modern TEX distribution for GNU/Linux

Michal Vlasák

MMTEX stands for “minimal modern TEX” and is a simple, small and legacy-free distribution of TEX for GNU/Linux. It has the form of a installable package, offers full functionality of OpTEX and plain LuaTEX formats, allows use of system fonts and resources in external TEXMF trees.

This article explains my motivation for creating it, describes some aspects of a distribution in general and how they are handled in MMTEX. My goal is to show that things can be done simply, and that TEX can integrate better into a Unix system and not be the odd one out.

Motivation

I find TEX Live huge and complicated. Its “full scheme” installation, which is the default, takes up about 7 GiB. Though with “minimal” (plain TEX only) scheme and no documentation you can get to about 50 MiB. I also think that it doesn’t fit very neatly into a Unix system. It isolates itself more or less in a single directory tree and doesn’t follow the hier(7) standard; as two examples, it doesn’t store configuration files in `/etc` and doesn’t permit read-only mount of `/usr`.

My end goal was to create something that:

- is a single package that can be installed using an operating system’s package manager,
- integrates well into a system (respecting filesystem hierarchy, using system fonts),
- includes only core functionality, but could easily be pointed to external TEXMF tree(s) with additional packages/files,
- doesn’t complicate things with legacy, dynamic regeneration of various files, ...

The intended users are those who want a small (34 MiB) but functional TEX system, one comparable to LATEX and its many packages. Because of its small size and ability to install using a package manager it can also be useful for Docker images or CI/CD scripts that need to set up a TEX environment.

Engine

Nowadays support for PDF output and OpenType fonts is a must. That leaves the choice of engine between LuaTEX and XƎTEX. LuaTEX was chosen, because it is extensible with Lua, has better support of micro-typographical extensions, and has integrated METAPOST in the form of `mplib`.

Compiling LuaTEX. Originally LuaTEX started with pdfTEX’s sources (mostly written in `WEB`), but was translated to C. LuaTEX is somewhat confusingly developed in a repository that is essentially a subset of TEX Live’s repository, but separate. TEX Live’s build infrastructure is based on GNU Autotools and is able to compile all of TEX Live and external libraries for *a lot* of platforms, but is very slow and does a lot of checks for things that have been standardized in C or POSIX for years.

After preparing replacements of build-time generated files it is however possible to compile LuaTEX in more or less a single call to the compiler — the only catch is `mplib`. It is written in CWEB, which itself is written in CWEB, so bootstrapping is needed.

External libraries. LuaTEX uses several external C libraries. The most prominent one is of course Lua, but it also uses, for example, `libpng` to handle PNG images. There are two choices for how to use a library — either compile its source into the binary (“statically link”) or on each run of the program find and load the compiled library file somewhere on the file system (“dynamically link”). The usual choice for most systems is dynamic linking — this allows reuse of a single library file for more programs (making updates easier) and saves disk space. It is a bit slower, because of the searching and loading.

The `libpng` and `zlib` libraries, for example, are often already present on systems or can easily be installed using a package manager. For these the dynamic linking approach is better. Other libraries have LuaTEX’s modifications (Lua) or are specific to TEX (Kpathsea) so sharing them would not be especially useful. These are statically linked.

Formats

The obvious choice of format for a minimal TEX would be plain TEX. Or rather its adaptation for LuaTEX which for example outputs to PDF by default. While it can be called minimal, it isn’t “modern”. Most users today expect a format to be able to easily create documents with numbered sections, tables of contents, bibliographies, hyperlinks, source code listings with syntax highlighting, etc. A recent format, based on LuaTEX, which provides these features, but still keeps plain’s simplicity is OpTEX. In a sense it is even more powerful than LATEX — where LATEX needs a package or an external binary, OpTEX has it built in.

Both formats are included in the distribution, OpTEX is the primary one, while plain is for now included more or less just to allow running `luatex` without getting an error about a missing format.

Finding files

LuaTEX uses the Kpathsea library for finding files. Kpathsea uses path specifications and variables similar to the Unix `PATH` environment variable, but differentiates between file types. For each file type it maintains one or more associated variable names, a list of possible suffixes and most importantly a calculated *search path* (directories separated by colons). For example `bib_format`'s variables are `BIBINPUTS` and `TEXBIB`, while the suffix list contains `.bib`.

In a simple case a search path is determined in one of three ways, in order of significance:

- value of associated variable set in the environment (that is, an environment variable),
- value of associated variable from a `texmf.cnf` configuration file,
- default value set at compilation time.

For finding `.cnf` files the same path searching mechanism is used; the variable is `TEXMFCNF`, but as it cannot be set from a configuration file, only the first and last way applies. All `texmf.cnf` files that are found are read. Order is important — earlier assignments in configuration files override later ones.

I set all the useful file types to have defaults that work without any configuration file, and respect standards like hier(7), TDS and XDG. For example, the search path for `.cnf` files is:

- `TEXMFDOTDIR` (more on this later),
- `~/.config/mmtex` (or more precisely its XDG equivalent),
- `/etc/mmtex`,

to allow local ("project"), user and system configuration files respectively.

In Kpathsea, default (compile-time) values for *search paths* can be set, but not *variables*. For this I created a patch that "injects" default values for a few variables, as if they were read from a configuration file.

`TEXMFDOTDIR` variable was inspired by TEX Live and is normally the current directory ("."), but is useful for temporary overrides on the command line, using environment variables. Every search path contains `TEXMFDOTDIR` as the first entry, even the one for `.cnf` files (allowing for project-specific settings).

`TEXMF` is the most important variable for MMTEX. It should contain roots of all TEXMF trees. It is supposed to be set by the user or system administrator at any level of configuration they need at the moment, and doesn't have a default value (preferences of users and system administrators vary widely).

Language support

Previous TEX engines had the limitation of being able to load hyphenation patterns only at format creation time — when running iniTEX. LuaTEX has no such limitation; by using Lua, it is possible to load hyphenation patterns at runtime.

Today virtually all hyphenation patterns and exceptions that have been used by TEX users are distributed in the `hyph-utf8` package. `hyph-utf8` also provides patterns and exceptions in UTF-8 encoded text files, which are preferred for LuaTEX.

TEX Live's approach is to provide hyphenation patterns and exceptions for each language in a separate package. Each package then hooks itself using the TEX Live `execute AddHyphen` directive. An example for French:

```
execute AddHyphen \
    name=french synonyms=patois,francais \
    lefthyphenmin=2 righthyphenmin=2 \
    file=loadhyph-fr.tex \
    file_patterns=hyph-fr.pat.txt \
    file_exceptions=
```

This information is also written to files used by ε-TEX's language mechanism, which is used by plain LuaTEX. This gets added to `language.def`:

```
\addlanguage{french}{loadhyph-fr.tex}{}{2}{2}
```

and this is written to `language.dat.lua`:

```
['french'] = {
    loader = 'loadhyph-fr.tex',
    lefthyphenmin = 2,
    righthyphenmin = 2,
    synonyms = { 'patois', 'francais' },
    patterns = 'hyph-fr.pat.txt',
    hyphenation = '', },
```

`etex.src` reads `language.def` at format creation time. Listed languages are registered and their hyphenation patterns loaded into the format. This enables their use later with `\uselanguage`.

In LuaTEX, it is discouraged to load patterns into the format, so the mechanism is changed by `hyph-utf8`'s own `etex.src`. Instead of loading each pattern or exception file on `\addlanguage`, the language is only registered and the files are loaded at the first `\uselanguage`. Both commands use Lua code in `luatex-hyphen.lua`, which uses information in `language.dat.lua` for handling synonyms and finding the names of pattern files.

In OpTEX the situation is simpler. It doesn't read `language.def` because it already has that information, but it still uses `luatex-hyphen.lua`.

To support all languages in `hyph-utf8`, MMTEX generates the files `language.def` and `language.dat.lua` from the `hyph-utf8` sources.

Fonts

To fully use the potential of LuaTEX, OpenType fonts should be used. These are the same fonts that are used by other programs, and as such some of them are already preinstalled on operating systems. And probably many more are additionally installed by users or administrators. To also not duplicate any effort with packaging of fonts, the distribution doesn't provide any OpenType fonts. The idea is to let users use the fonts they already have or can get on their system, as well as the fonts they have in their TEXMF tree(s). For example Latin Modern, the GUST e-foundry adaptation of Computer Modern which includes OpenType, is available as `fonts-lmodern` on Debian-based systems and `otf-latin-modern` on Arch Linux.

8-bit fonts. Only 8-bit fonts can be preloaded into a TEX format. Both OpTEX and plain LuaTEX do this. To support this, MMTEX includes a minimal set of Type 1 fonts and their respective metric and encoding files. A `pdftex.map` file is needed, as it is used to map names of TFM metric files to font names and font files, with optional reencodings. This file contains lines like:

```
cmr5 CMR5 <cmr5.pfb
ec-lmr5 LMRoman5-Regular <lm-ec.enc <lmr5.pfb
```

The first line connects the `cmr5.tfm` font metric file, the `cmr5.pfb` Type 1 font and the `CMR5` font name inside the `.pfb`. (CMR5 stands for Computer Modern Roman in 5 point optical size). The second line is similar, but additionally refers to a so-called encoding vector stored in file `lm-ec.enc`. This is necessary because `lmr5.pfb` contains many glyphs, while TEX can use only 256 of them and expects the order to correspond with `ec-lmr5.tfm`, which contains metric information for those selected 256 glyphs. In this particular case the Cork ("EC") encoding (set of glyphs) is used.

Engines only read one `pdftex.map` file, but each font package usually provides one or more `.map` files. This is why an aggregate `pdftex.map` is usually generated (in TEX Live using the `updmap` script). As MMTEX supports only a limited number of Type 1 fonts, a minimal `pdftex.map` was created by hand.

OpenType fonts. In order to handle OpenType fonts Lua code is needed. `luaotfload` is included for this purpose as it is already used internally by OpTEX. It can also be used from plain LuaTEX, with `\input luaotfload.sty`.

`luaotfload` is able to find all system fonts, because it reads `fontconfig`'s configuration. Therefore, there is no need to set the `OSFONTDIR` variable. The standard TDS directories for font files also work: `$TEXMF/fonts/{opentype,truetype}`.

MetaPost

METAPOST is integrated into LuaTEX as `mplib` and available via a Lua interface. `luamplib` adapts the code from ConTEXt for plain (and LATEX), making it possible to use METAPOST in a `.tex` file. To use it, `\input luamplib.sty`.

`mplib` proved to be useful even as a METAFONT replacement. "Ralph Smith's Formal Script" font (required by OpTEX) doesn't have prebuilt TFM files on CTAN. Normally one would use METAFONT to generate the metrics, but with a few lines of Lua `mplib` can, just like METAPOST, use the `mfplain.mp` format to function as METAFONT and do the job.

Implementation of MMTEX

MMTEX itself is a few supporting files and a script called `build` which contains instructions for building MMTEX to a given directory. The script in this form allows a wrapper that packs together the directory and some metadata to create an installable package. The included `package-builder` script demonstrates this, and as of now can create packages in Debian's `.deb`, Arch Linux's `pkg` and classic tarball (`.tar.gz`) formats.

The sources (which are merely the build logic), documentation, and prebuilt packages are available at `https://github.com/vlasakm/mmtex`.

The result

The resulting package, installed, takes up 34 MiB (around 15 MiB compressed). Most of this is Type 1 fonts (11 MiB), the `luatex` binary (6.5 MiB), data related to Unicode codepoints (about 5 MiB) and hyphenation patterns/exceptions (3.2 MiB).

Not included are macro source files (`.dtx`) and package documentation , both of which are of course available on CTAN. Documentation is also easily accessible on `https://texdoc.net`.

At present there is no support for getting or managing packages from CTAN — MMTEX expects to be pointed to already-prepared TEXMF trees. OpTEX hopefully provides enough functionality to not require a large number of other macros.

◇ Michal Vlasák
Proboštov, Czech Republic
lahcim8 (at) gmail dot com