ConTEXt in TEX Live 2023

Hans Hagen

Starting with TEX Live 2023 the default ConTEXt distribution is LMTX, a follow up on MkIV, running on top of the LuaMetaTEX engine instead of LuaTEX. Already for a long time the MkII version used with pdfTEX, XƎTEX and Aleph has been frozen and most users moved on from MkIV to LMTX (a more distinctive tag for what internally is version MkXL).

In principle one can argue that we now have three versions of ConTEXt and there can be the impression that they are very different. However, although MkXL can do more than MkIV which can do more than MkII, the user interface hasn't changed that much and old functionality is available in newer versions. Of course some old features make no sense in newer variants, like eight-bit font encodings in an OpenType font realm and input encodings when one uses UTF, although we still support input encodings a.k.a. regimes. When we started using the Mk* suffixes the main reason was that we had to distinguish files and the official TEX distribution doesn't permit duplicate file names. Using a distinctive suffix also makes it possible to treat files differently.

Table 1 shows major aspects of the different ConTEXt versions. The 'template' files listed in the table are a mix of TEX and Lua and originate in the early days of MkIV; basically, they are a wink to active server pages. With 'arguments', we refer to files that accept named macro arguments which means that they need to be preprocessed. That started as a proof of concept but some core files are defined that way. Users will normally just use a `.tex` file.

The Lua files in the code base have the suffix `lua`, or when meant for LuaMetaTEX that uses a newer Lua engine they can have the suffix `lmt`. There can also be `lfg` (font goodies) and `llg` (language goodies) plus byte-compiled files with various suffixes but these are normally not seen by users. We leave it at that.

So, while TEX Live 2022 installed MkII and MkIV, TEX Live 2023 installs MkIV and LMTX. Therefore the most significant upgrade is in the engine that is used by default: LuaMetaTEX instead of LuaTEX. The MkII files are no longer installed so we don't need pdfTEX.

So how did we end up here? Initially the idea was that, because LuaTEX is basically frozen, LuaMetaTEX would be the engine that we conduct experiments with and from which occasionally we could backport code to LuaTEX. However it soon became clear that this would not work out well so backporting is off the table now. Just for the record: the project started years ago so we're not talking about something experimental here. There have been articles in *TUGboat* about what we've been doing over the years.

One of the first decisions I made when starting with LuaMetaTEX was to remove the built-in backend, which then meant also removing the bitmap image inclusion code. That made us get rid of dependencies on external libraries. In fact, a proof-of-concept experimental variant didn't use the built-in backend at all. The font loading code could be removed as well because that was not used in MkIV either. In MkIV we also don't use the `kpse` library for managing files so that code could be dropped from the engine tool; it can be loaded as so-called optional library if needed but I'll not discuss that here. If you look at what happens with the LuaTEX code base, you'll notice that updating libraries happens frequently and that is not a burden that we want to impose on users, especially because it also can involve updating build-related files. Another advantage of not using them is that the code base remains small.

A direct consequence of all this was that the build process became much more efficient and less complex. A fast compilation (seconds instead of minutes) meant that more drastic experiments became possible, like most recently an upgrade of the math subsystem. All this, combined with an overhaul of

suffix	engine	template	arguments	main file
MkII	pdfTEX, XƎTEX, Aleph			`context.mkii`
MkIV	LuaTEX, LuaJITTEX, LuaMetaTEX			`context.mkiv`
MkVI	idem		yes	
MkIX	idem	yes		
MkXI	idem	yes	yes	
MkXL	LuaMetaTEX			`context.mkxl`
MkLX	idem		yes	

Table 1: Major ConTEXt versions.

the code base, both the TeX and MetaPost part, meant that backporting was no longer reasonable. Being freed from the constraint that other macro packages might use LuaMetaTeX in turn resulted in more drastic experiments and adding features that had been on our wish list for decades. Another side effect was that we could easily compile native Windows binaries and immediately support transitions to ARM-based hardware.

Instead of "backporting after experimenting", a leading motive became "fundamentally move forward" while at the same time tightening the relation between ConTeXt and the engine: the engine code became part of the distribution so that users can compile themselves, which fits perfectly in the paradigm (and demands) of distributing all the source code, even that of the engine. There is also less danger that patches on behalf of other usage interferes with stable support for ConTeXt. A specific installation is now more or less long-term stable by design because it no longer depends on binaries and/or libraries being provided for a specific platform and operating system version. Of course installers and TeX Live do provide the binaries, so users aren't forced to worry about it, but they can move along with a system update by recompiling an old, and for their purpose, frozen ConTeXt code base.

An unofficial objective (or challenge) became that the accumulated source stays around 12 MB uncompressed, (compressed a bit over 2 MB) and the binary around 3 MB so that we could use the engine as an efficient Lua runner as well as a launcher stub, thereby removing yet another dependency. That way the official ConTeXt distribution didn't grow much in size. A bonus is that we now use the same setup for all operating systems. It also opened up the possibility of a exceptionally small installation with all bells and whistles included. Another nice side effect, combined with automatic compilation on the compile farm, makes that we can provide installations that reflect the latest state of affairs: a recent binary combined with the latest ConTeXt. As a result, most users quickly went for LMTX instead of MkIV.

In the code base we avoid dependencies on specific platforms but there are a few cases where the code for Windows and UNIX differs. However, the functionality should be the same. A good test is that for Windows we can compile with mingw (cross-compilation), MSVC (native) and clang (native); that order is also the order of runtime performance. The native MSVC binary is the smallest but users probably don't care. In any case, it is nice to have a fallback plan in place. The code is all in C; the MetaPost code is converted from CWEB into C using a Lua script but we also ship the resulting C code. The code base provides a couple of CMake files and comes with a trivial build script.

When I say that there are no libraries used, I mean external libraries. We do use code from elsewhere: adapted `avl` as well as `decnumber` (for the MetaPost library), adapted `hjn` (hyphenation), `miniz` (zip compression), `pplib` (for loading PDF files), `libcerf` (to complement other math library support, but it might be dropped), and `mimalloc` for memory management. However all the code is in the LuaMetaTeX code base and only updated after checking what changed. The most important library originating elsewhere is of course Lua: we use the latest and greatest (currently) 5.4 release. We kept the `socket` library but it might be dropped or replaced at some point. In addition there is a subsystem for dynamically loading libraries; the main reason for that being that I needed `zint` for barcodes, interfaces to sql databases, a bunch of compression libraries, etc. But all that is tagged *optional* and ConTeXt will never depend on it. There are no consequences for compilation either because we don't need the header files. The glue code is very minimalistic and most work gets delegated to Lua.

Initially, because the backend is written in Lua, there was a drop in performance of some 15% but that was stepwise compensated by gains in performance in the engine and additional or improved functionality. The ConTeXt code base is rather optimized so there was little to gain there, apart from using new features. Existing primitive support could also be done a bit more efficiently; it helps if one knows where potential bottlenecks are. Therefore, in the meantime an LMTX run can be quite a bit faster than a MkIV run and it can even outperform a LuaJITTeX run. In practice, the difference between an eight-bit MkII run using the eight-bit pdfTeX engine and a 32-bit LuaMetaTeX run with LMTX can be neglected, definitely on more complex documents. I never get complaints about performance from ConTeXt users, so it might be a minor concern.

So what are the main differences in the installation? If you really want to experience it you should use the standard installation. Currently the small installer is the engine that synchronizes the installation over the net and, assuming a reasonable internet connection, that takes little time. The installation is relatively small, and many of the bytes used are for the documentation. Updates are done by transferring only the changed files. The TeX Live installation is a bit larger because it shares for instance fonts with the main installation and these come with resources used by other macro packages. Both installations bring MkIV as well as LMTX and therefore provide LuaTEX as well as LuaMetaTEX. However, a MkIV run is now managed by LuaMetaTEX because we use that engine for the runner. The MkII code is no longer in TEX Live but is in the repositories and used to test and compare with pdfTEX. It just works.

The number of binaries and stubs is reduced to a minimum:

`luametatex`	combined TEX, MetaPost, Lua engines
`mtxrun`	script runner, binary
`context`	ConTEXt runner, binary
`mtxrun.lua`	script runner, Lua code
`context.lua`	loader for ConTEXt runner
`luatex`	the good old ancestor

All of these programs are in the ConTEXt distribution directory `tex/texmf-`⟨*platform*⟩`/`. In addition, `context` and `mtxrun` are symlinks to the `luametatex` binary, where possible.

So, the `context` command runs `luametatex`, but loads the Lua file with the same name which in turn will locate the ConTEXt management script (`mtx-context`) in the TEX tree and run it. The same is true for `mtxrun`: it is a binary (link) that loads the script in (this time) the same path and then can perform numerous tasks. For instance, identifying the installed fonts so that they can be accessed by name is done with:

```
mtxrun --script font --reload
```

Where in MkII we had stubs for various utility scripts, already in MkIV we went for a generic runner and a bit more keying. It's not like these scripts are used a lot and by avoiding shortcuts there is also little danger for a mixup with the ever-growing list of other scripts in TEX Live or commands that the operating system provides.

The LuaTEX binary is optional and only needed if a user also wants to process MkIV files. There are no shell scripts used for launching. The two main calls used by users are:

```
context foo.tex
context --luatex foo.tex
```

A user has only to make sure that the binaries are in the path specification. When you run from an editor, the next command does the work:

```
mtxrun --autogenerate --script context ⟨filename⟩
```

with ⟨*filename*⟩ being an editor-specific placeholder. Like other engines, LuaMetaTEX (and ConTEXt) needs a file database and format file, and although it should generate these automatically you can make them with:

```
mtxrun --generate
context --make
```

The rest of the installation is similar to what we always had and is TDS compliant. The source code of LuaMetaTEX is included in the distribution itself (which nicely fulfills the requirements) but can also be found at `github.com/contextgarden/luametatex`.

There are also some optional libraries there but ConTEXt works fine without them. The official latest distribution of ConTEXt itself is:

```
github.com/contextgarden/context
github.com/contextgarden/context-distribution-fonts
```

We see users grab fonts from the Internet and play with them. They can install additional fonts in `tex/texmf-fonts/data/`⟨*vendor*⟩. Project-specific files can be collected in `tex/texmf-project/tex/context/user/`⟨*project*⟩. These directories are not touched by installations and can easily be copied or shared between different installations. After adding files to the tree `mtxrun --generate` will update the file database.

In the distribution there are plenty of documents that describe how LuaMetaTEX with LMTX differs from MkIV with LuaTEX: new primitives, macro extensions, more granular math rendering, improved memory management, new (or extended) (rendering) concepts, more MetaPost features; most is covered in one way or another, and much is already applied in the ConTEXt source code. After all, it took a few years before we arrived here so you can expect substantial refactoring of the engine as well as the code base, and therefore eventually there is (and will be) more than in MkIV.

When you compare a ConTEXt installation with what is needed for other macro packages you will notice a few differences. One concerns the way TEX is launched. An engine starts with a blank slate but can be populated with a so-called format file that is basically a memory dump of a preloaded macro package. So, the original way to process a file is to pass a format filename to the engine. In order to avoid that a trick is used: when an engine (or symlink/stub to it) is launched by its format name, the loading happens automatically. So, for instance `pdflatex` is actually an equivalent for starting pdfTEX with the format file `pdflatex.fmt` while `latex` is pdfTEX with another format file (`latex.fmt`) starting up in DVI mode. And, as there are many engines, a specific macro package can have many such combinations of its name and engine.

In ConTEXt we don't do it that way. One reason is that we never distinguished between backends: MkII uses an abstract backend layer and load driver files at runtime (it was one of the reasons why we could support Acrobat as soon as it showed up, because we already supported the now obsolete but quite nice DVIWINDO viewer). And that model hasn't changed much as we moved on. Because we use a runner, we also don't need to distinguish between engines: all formats have the same name but sit on an engine subpath in the TEX tree. Anyway, this already removes quite some formats. On the other hand, ConTEXt can be run with different language specific user interfaces which means that instead of just `context.fmt` we have `cont-en.fmt` and possibly more, like `cont-nl.fmt`. So that can increase the number again but by default only the English interface is installed. As a side note: where with MkII we needed to generate MetaPost mem files, with its descendants having MPlib we load the (actually quite a bit of) MetaPost code at runtime.[1]

In addition to a format file, for the LuaTEX and LuaMetaTEX engine we also have a (small) Lua loader alongside the format file. All this is handled by the runner, also because we provide extensive command line features, and therefore of no concern to users and package maintainers. However, it does make integrating ConTEXt in for instance TEX Live different from other macro packages and thereby puts an extra burden on the TEX Live team. Here I want to thank the team for making it possible to move forward this way, in spite of this rather different approach. Hopefully a LuaMetaTEX integration is a bit easier in the long run because we no longer have different stubs per platform and at least the binary part now has no dependencies and only has a handful of files.

For those new to ConTEXt or those who want to try it in TEX Live 2023 there is not much difference between the versions. However, MkIV is now frozen and new functionality only gets added to LMTX. Of course we could backport some but with most users already having moved on, it makes no sense. Just as we keep MkII around for testing with pdfTEX, we also keep MkIV alive for testing with LuaTEX. Maybe in a couple of years MkIV will go the same route as MkII: ending up in the archives as an optional installation.

◇ Hans Hagen
Pragma ADE

[1] Occasionally I do experiments with loading the TEX format code at runtime, but at this moment the difference in startup time of about one second (assuming files are cached) is too large and running over networks will be less fun, so the format file will stay. The time involved in loading MetaPost can be brought down but for now I leave it as it is.

Preserving the math class of variables

Hans Hagen

If there is one thing that OpenType math has made clear, it's that we have lots of alphabets. It is customary in a TEX document to key in regular (ASCII) letters and expect them to become for instance math italic, bold upright, script or whatever.

One way to do this is to relate a character (directly or by name) to a specific slot in a font assigned to a so-called math family, which groups text, script and scriptscript sizes. Here are a couple ways to do this, using the Unicode `\Umathcode` primitive:

```
\Umathcode‘a = "0 "9 ‘a
\Umathcode‘a = "0 "5 "1D44E
```

In the first line we map the input character `a` (the first `‘a`) to the glyph slot of `‘a` (the second one; that is, 97) in family 9. In the second line, the input `a` is mapped to the Unicode math italic alphabet's a, using family 5. The `"0` in both lines is the math class, in this case specifying an "ordinary" character.

Switching families can be done directly, although more usually it is wrapped in a command:

```
$ a + {\fam"9 a} + {\fam"5 a} $
```

For our next example, we take a colon from family zero (`"0`) and assign it class 6 (`"6`) which means that it will get punctuation spacing (like `\Colon`):

```
\Umathchardef\foo "6 "0 ‘: % punct
```

In the following line we do the same but with class 7, which is "variable", meaning TEX uses the current family, as stored in the `\fam` primitive parameter.

```
\Umathchardef\foo "7 "0 ‘: % ord
```

Doing this, we lose the prior class value (3), so we end up with ordinary (which normally means no) spacing. In LuaTEX (>1.15.1) we can now preserve the class by declaring and using a special "variable" family instead:

```
\variablefam"24
\Umathchardef\foo "6 "24 123 % punct
```

When a character has family `\variablefam` assigned, it will get the current `\fam` value and the class can remain 3, as specified.

This is a relatively cheap extension which we prototyped in LuaMetaTEX and backported to LuaTEX. We don't use this in ConTEXt (just to warn its users) but it might be handy in other macro packages.

◇ Hans Hagen
Pragma ADE

doi.org/10.47397/tb/44-1/tb136hagen-classes