## ConTEXt LMTX

Hans Hagen

### 1 Introduction

More than decade after introducing the ConTEXt version for LuaTEX, flagged MkIV (the version known as MkII ran under more traditional TEX engines), it is time for something new. As MkVI, MkIX and MkXI are already taken for special variants of MkIV files a new acronym was coined: LMTX. Such a sequence of letters must sound somewhat fancy but it actually has a meaning, or better: it has several. One is that it stands for LuaMetaTEX, an indication of an engine to be used. But one can also think of the components that make up ConTEXt code: Lua, MetaPost, TEX, and XML. But once it is clear to a user what (s)he is dealing with, it can also indicate a "Lean and Mean TEX eXperience". But, as with MkIV, eventually it will become just a tag.

So, with this out of the way, the next question is, what do we mean by LuaMetaTEX? The term MetaTEX first surfaced at a ConTEXt meeting but it is actually a variant of what Taco and I always thought of when we discussed LuaTEX: an engine that hosts not only TEX but also MetaPost, which we all like quite a lot. But, as Lua is now an integral part of this adventure, it has been glued to more than just the engine name.

In the next sections it will be clear what we're actually talking about and how this relates to ConTEXt LMTX. But let me first stress that after a decade of usage, I'm convinced that the three main components fit nicely together: TEX has a good track record of usability and stability; MetaPost is a nice graphical description language, efficient and very precise, and Lua is, in my opinion, the nicest scripting language around, minimal, rooted in academia and developed in a steady and careful way. It makes for a hard to beat combination of languages and functionality.

### 2 Using ConTEXt

The average ConTEXt user (MkIV) needs only the LuaTEX engine, a bunch of macros, fonts and hyphenation patterns. One can install ConTEXt from the TEX Live distribution (or via a distribution's software package) or directly from the ConTEXt garden. The first ships the yearly, so-called 'current' release; the second also provides intermediate 'beta' releases. In both cases the number of installed bytes is reasonable, especially when one compares it to many programs or packages. In fact, relative to many programs that come with a computer ecosystem these days, a TEX installation is no longer that large.

Most users will use ConTEXt from within a text editor, normally by hitting a key to process; others might run it on the command line. Then there are those who run it as part of a complex workflow where no one sees it being run at all. When you run LuaTEX with ConTEXt, there is the usual log data flying to a console as well as some short delay time involved to get the result. But that is the game so there is nothing to worry about: edit, run, wait and watch.

On a multi-core machine, only one CPU core will be taken and, depending on the document, a reasonable amount of memory. As ConTEXt evolves we try to limit its use of resources. If we take almost any browser as benchmark then TEX is cheap: its memory consumption doesn't slowly grow to persistent gigabytes, there is no excessive (unnoticed but not neglectable) writing to disk, and when one is just editing a document it will not suddenly take CPU cycles. We try not to turn the ConTEXt+LuaTEX combination into bloatware.

If you listen to what is discussed between the lines at the ConTEXt meeting you will notice that some use this rendering system as a sort of appliance: it does the job without the end user of the document ever realizing what is involved. Using some virtual machine in this case is quite normal. When ConTEXt is running on a server or background system we need to keep in mind that performance is not required to improve that much and that low power solutions are considered more often. This also means that we must try to lower the memory footprint as well as the need for CPU cycles.

There are many cases where the system is used to generate (few) page documents as part of a larger workflow (or program). One example is creating a PDF file from a MetaPost graphic (with TEX) that gets converted to SVG and then ends up in a web page (part of a status/input screen). It's hard to beat MetaPost in terms of quality and cost effectiveness. For this a lean and mean and well contained setup is needed. But, in order to be permitted to use such a subsystem one could be asked to prove that what's underneath is not some complex, hard to maintain component. It being open source is not enough.

Users find it hard to convince employers to use a TEX system. It is seen as large, is considered old, doesn't always fit in. Therefore, contrary to what one expects, expensive, less robust and less future safe solutions are chosen. How we can help users addressing this problem was also a topic discussed at the last ConTEXt meeting.

All these aspects (of usage) have led to what I present now as ConTEXt LMTX, which started mid-

2018 and is expected to be stable mid-2019, after a year of intense experimenting and development. This is in time for the 2019 ConTEXt meeting where we can pick up discussions about how to make sure TEX is a valid candidate for rendering (so far as that is still needed in the browser dominated arena).

## 3 Packaging ConTEXt

We have now arrived at a brief summary of what ConTEXt LMTX actually is, but let's stress that for the average user it is what they already have: ConTEXt MkIV using LuaTEX. In fact, we will continue to ship what has been there for years alongside LMTX so that users can test and we can fix bugs. Some parts of the code base already have LMTX specific branches but so far (most) users never enter these.

- We use a lean and mean engine: LuaMetaTEX identifying itself as LuaTEX 2.0 (we count from there). The binary is some mere 3 MB which is much smaller than stock LuaTEX.[1] The size matters because the engine is also a Lua processor and used as such.
- The LuaMetaTEX code base is also relatively small and is now part of the ConTEXt distribution. This means that when one downloads the archive or installs ConTEXt, the source is included! One gets the whole package. The 12 MB source tree compresses to around 2 MB.[2]
- If needed the user can compile the program. The build is relatively simple with essentially no dependencies, very few requirements, and all files that are needed are included. The average user will not compile but it adds to the idea that the user is independent and that, when ConTEXt is used as a component, all batteries are included.[3] With the sources in the distribution, users on non-standard systems can easily bootstrap a working system.
- Where LuaTEX depends on a few external libraries, LuaMetaTEX goes with the absolute minimum as it is what the name says: a core TEX engine, the MetaPost machinery, and Lua.[4] The few libraries that we ship are part of the source tree and their interfaces are unlikely to change.
- There is just one program: LuaMetaTEX. We use Lua 5.4 and no longer need LuaJIT as it lags behind and has no real significant performance advantages. There are no extra binaries needed, as this one program also serves as a stub. The first experiences have demonstrated that Lua 5.4 is somewhat faster than its predecessors. We plan to use the more efficient generational garbage collector once it becomes stable.
- When it comes to installing ConTEXt, the engine is also the installer. Instead of using `rsync` we use `http`. An initial install can take a little time, but updates much less. Once the installer is unzipped there are no dependencies on any other programs. The small size of the binary facilitates such usage.
- The installation only has the files needed for MkIV. Of course there is still the archive with everything, but there is no need to install MkII files and resources when only LuaMetaTEX is used. At some point the installer will allow the installation of both MkII and MkIV.
- The MkIV codebase is aware of the engine and triggers LMTX mode accordingly. It will use (a few) different files when it runs on top of LuaTEX or LuaMetaTEX. There might be a point in time when we make a more rigorous split.

## 4 Fundamental changes

A user now thinking "So what?" deserves some more explanation about what has been changed under the hood. Part of the answer relates to the shrunken binary. How that happened is discussed in a document that kept track of the process and decisions but probably is only of interest for a few. A short summary follows.

At some point last year I realized that I was coming up with solutions where LuaTEX was actually working a bit against me. One reason for that is that we had to become stable at some point and could not change it fundamentally any longer. The main reason for that was that other macro packages are also using it: even a trivial change (say in the log) can spam your mailbox with complaints. At the same time it started annoying me that we had to carry around quite a bit of code that ConTEXt doesn't need or use at all. In addition to that, there were some dependencies on foreign code that resulted in occasional (enforced) updates of the rather complex source tree and build structure. The switch to a new PDF introspection library had demonstrated that dependencies could simply be made much less troublesome. But, as TEX is basically just juggling bytes I wondered if it could be done even better (hint: in Lua).

Already a while ago ConTEXt started using its own OpenType font loader, written in Lua instead of the one derived from FontForge. But actually that built-in loader code was the last to go as bits and

[1] Of course we still pay attention to LuaTEX, but there we have more or less frozen the feature set in order to make it a future-safe engine.

[2] This will be at the formal release at the 2019 meeting.

[3] For now, I directly handle the Windows and Linux binaries and Alan Braslau takes care of the OSX and FreeBSD binaries. We have decided on providing 64 bit binaries on these systems that we actively use and in the future they will be generated on the compile farm.

[4] The only shared libraries that are referenced are libc, libm, and libdl.

pieces were still hooked into the different parts of the code base, for instance in the PDF backend.[5]

So, the first thing to get rid of was image inclusion library code. Much of the inclusion was already under strict ConTEXt control anyway. Of course we still can include images but ConTEXt does it entirely under Lua.[6]

Next to go was font embedding. In ConTEXt we already did most of that in Lua, so this kick-out was not that dramatic. As a natural followup, generating the page stream was also delegated to Lua. For the record: in all these stages I had a hybrid code base, meaning that I could continue to do the same in LuaTEX. Only at some point the diversion was too large for comfort, and I switched to dedicated code for each engine. Also, the regular ConTEXt code had to keep working so every (even temporary) change had to be done very carefully.

Once the built-in backend was reduced that way, I decided to ship out the whole page via Lua. Keep in mind that in ConTEXt MkIV we always only needed part of the backend: we never depended on any of the extensions supported by the engine and did most PDF-specific features differently already. But for macros using these extensions, it's one of the components that for sure will suffer in performance from being written in Lua instead of C code.

We now no longer had any need for the code in the built-in font loader: we didn't use the loading code but the backend still had used some of its magic for inclusion. So, out went the loader. When I realized that TEX needs only a little information to support what we call base mode, I decided that we could also replace the TFM loader by a Lua loader. We already had to deal with virtual fonts in the new backend code anyway. So, basically most font related code is now gone: only a little is passed to the engine now, that which is needed to do the typesetting.

With the backend reduced to a minimum, it was then a small step to removing it altogether. I didn't feel too guilty about this because officially TEX has no backend: it's an extension. So, the `img` and `pdf` libraries are also gone. With the whole PDF file now generated directly by ConTEXt it was time to deal with extensions. Stepwise these were removed too. In the end we only had a generic so-called 'whatsit' left. The impact on the ConTEXt code is actually not that large: the biggest complication is that we need to support both stock LuaTEX and LuaMetaTEX, so figuring out a way to do that (stepwise as we had a transition) took a while. When doing something like this one should not be afraid to write temporary code and later take it out.[7]

I used these opportunities to improve the readability of the uncompressed PDF output a bit (some was already done in LuaTEX). It must be noted that the final virtual font handling happens in the backend (TEX only cares about dimensions) so this can open up a whole new world of possibilities. In principle the produced code can be made a bit more efficient. I must admit that I always treated much of the font code in the backend as a black box. Reverse engineering such code is not my forte (and no fun either) so I tend to just figure it out myself using the formal specifications. That is also more fun.

I didn't mention yet that parallel to this process some more cleanup happened. All code files were touched; where possible, header files were introduced. And every (even small) step was tested by processing documents. In some places the code could be made more efficient because we had less interwoven code as a side effect of removing the backend font and image related code. I also cleaned up a bit of the Lua interface and more might happen there. Some libraries lost features due to the conceptual changes. Also, libraries like `slunicode` were removed as we never truly needed them. In ConTEXt we already used adapted and optimized socket interfaces so we also no longer preload the old ones that come with the socket libraries. There are lots of such details, which is why it took many months to reach this state.

There are fewer command line options and the startup is somewhat streamlined. As in ConTEXt we already are (mostly) `kpse` compatible but entirely in Lua, as that library was removed too. This affected quite a bit of code, however, because the backend is outsourced, also a lot of file handling! Basically only TEX and Lua files are now seen by the frontend. And dealing with system calls was already done in Lua. We don't need much on top of what we accumulated in Lua for over a decade.

One can wonder if we're still talking TEX and the answer (at least for me) is yes we are. Original TEX has only a DVI backend and DVI is nothing more than a simple page description (I can cook up one if I need to). One needs an external program to produce something useful from DVI. Sure, pdfTEX grew to add a PDF backend but that again is an extension, and although LuaTEX separates the code better than

[5] Already some time ago I have made sure that this part is not ever used by ConTEXt, which made me confident that at some point the entire library could be removed.

[6] We still keep the PDF introspection library available as it has some advantages over using Lua, but I have developed a somewhat limited Lua alternative to play with; maybe a possible future variant.

[7] At some point I will add a DVI backend, just for good old times.

its ancestor, initializations for instance still mess up the code. The only place where extensions and built-in standard functionality, reflected in primitives, overlap is in writing to files. Clearly we do need to support that. However, along with some other (LuaTEX) primitives, the backend-related ones are gone. But ... we can simply implement them using Lua so that a macro package still sees these primitives. Nowhere is it mandated that a 'primitive' should be hardcoded in the engine.

In fact, one reason for going this route is that it is a way to come closer to the original, even as we have a few more primitives (ε-TEX as well as LuaTEX). But what about directions, those coming from Aleph (Omega)? It is a fact that in spite of attempts to deal with all these directions, only a few made sense, and LuaTEX ended up with only four. Of these four, only left-to-right and right-to-left ever worked well. I cannot imagine someone using the vertical ones as they are hard to control. Therefore, as soon as the backend was gone, I decided to keep only the two horizontal directions. Yet, in order to still support vertical rendering boxes got official offsets and orientations, at the TEX level. Of course the backend is free to interpret this. This might find its way back to stock LuaTEX but only after I'm satisfied for some time. So, more about this in another article. I kept only the new numeric direction specifiers.

With all this done, simplifying the binary build was on the agenda. This was not trivial. In retrospect I should just have started new, but because of all those dependencies it made more sense to stepwise strip the process to get an idea of what was happening and why. In the end, when it was determined to be sort of impossible to go much smaller, I decided to quit that and just write a bunch of CMake files. The retrospect relates to the fact that this took me a day to figure out for Windows, Linux, OSX and ARM. A nice side effect was that the LuaMetaTEX engine compiles in about 30 seconds on my about eight-year-old laptop, which suddenly got an extended lifetime (and about 15 seconds for Alan on an almost four-year-old Macbook with an SSD).

## 5 The roadmap

The roadmap is as follows. Core development took place in Fall 2018 and Spring 2019. On April 1 there was a version (2.00) suitable for use so users could start playing with this variant. At that time, a prototype of the new installer was also ready and tested by some other developers. After this first release I can start optimizing components in ConTEXt that are currently sort-of hybrid due to the fact that code has to run on both engines, but that code was not yet distributed. Around the ConTEXt meeting in Fall 2019 documentation should be available at which time the LuaMetaTEX source code will also become part of the distribution. Hopefully binaries will then be generated using the compile farm. After that the ConTEXt code base will become more and more tuned for LMTX. This all has to happen in such a way that no matter what engine is used, ConTEXt works as expected. In the process we need to make sure that users who (for some reason) use obsolete low level libraries are served by proper replacements.

## 6 Summary

So what we have now is a lean and mean engine. What are the consequences for users? They get a smaller but all-inclusive package. The lack of single backend and dependencies is also more future proof. Although a native backend performs slightly better for simple text-only documents, any more complex document is processed as fast or faster, and we can hope to gain a bit more over time. For instance, processing a LuaTEX manual with current LuaTEX and LMTX code takes 13.0 seconds, while using the native backend takes 12.6. But LuaMetaTEX with LMTX currently needs 11.7 seconds, so while we lost some on employing more Lua we gained a bit in the engine. The manual still can be processed a bit faster using LuaJITTEX, but for other documents the new setup can actually beat that variant by a wide margin. I will not go into details of how this happens because it is probably rather ConTEXt specific. At any rate, I always try to make sure that whatever I change deep down in ConTEXt, performance is not hit.

I'm quite satisfied that we now have a clean code base, fast compilation, a bit more Knuthian (less loaded) engine, and that we can distribute all in one package without an increase in size. Combined with the new installer this is quite a step forward in ConTEXt development, at least for me.

For me, furthermore, the main effect is that I have more freedom for experimenting and prototyping features that could be fed back into LuaTEX. I also hope that eventually this machinery is well suited for low power computers (and servers) with limited memory. I will probably report more on all this at another time.

I'd like to thank Alan Braslau for his support in this project, his patient testing, and belief in the future. He also made this article a bit more readable. We love these three languages and have lots of plans!

◇ Hans Hagen
http://pragma-ade.com