LuaMetaTEX

Where do we stand?

Hans Hagen

When it started

About three years ago, we conceived the idea of LuaMetaTEX. It was presented to members at the 2018 meeting and was broadly accepted. In early 2019, the first beta version was released and by the 2019 meeting, the first official version was presented.

Around the time of the 2020 meeting, we had more-or-less arrived at what I had in mind. By the 2021 meeting, I am expecting the code to be stable and the repositories to be set up. At the 2022 meeting we can make the official transition from MkIV to LMTX.

Some new options are only enabled in my local `cont-exp.tex` file. Knowing that Wolfgang keeps an eye on all these changes makes me more daring. We aim for less (but more efficient) macro code that, on average, looks better.

Why it started

There was an increasing pressure for a stable LuaTEX: no more changes to the interfaces, no more extensions.

One can run into interesting comments on the web (as usual), like

- The LuaTEX program has 'many bugs'.
- The LuaTEX manual is bad.
- The LuaTEX program is too slow to be useful.
- The LuaTEX program will never end up in distributions.
- The LuaTEX project is funded and developed in a commercial setting.

I won't comment on how I read these (demotivating) comments because they often says more about the writers and their attitudes than about LuaTEX.

It also looks like non-ConTEXt users are charmed by LuaTEX, and the more they code, the more we need to freeze the code base.

So, hopefully, the LuaMetaTEX development does not interfere badly with developments outside the ConTEXt community.

The development

The summary on the next pages is partial, but more information can be found in articles and documents that come with the distribution.

LuaTEX started out as cweb code ... this eventually became just C ... which in LuaMetaTEX has been detached from the (complex) infrastructure.

The basic idea was to keep only the core of TEX, dispensing with, for instance, font loading, file handling and the backend. As a consequence the code has been reorganized.

I experimented a lot without bothering about usage elsewhere and so far, I like the results.

The ConTEXt distribution will at some point ship with the source.

File handling

All file handling goes via Lua, including the read and write related primitives. The same is true for terminal (console) handling.

Some parts (namely the file writing code) were actually kind of extension code in TEX and partly a system dependency.

The ε-TEX pseudo file `\scantokens` primitive uses the same mechanism as Lua does.

The macro machinery

There are extensions to the way macro arguments are handled, allowing for less clumsy macros.

There are extra 'if' tests, making for nicer macros.

'Else' branches in conditions can be collapsed using `\orelse` and `\orunless` which makes the low-level code cleaner.

Tracing gives more detail about node properties and also shows their attributes.

Some new data carriers have been added that can be played with from Lua.

Macros can efficiently be 'frozen' (new) and 'protected' (redone), and the concepts 'long' and 'outer' are gone.[1]

Saving and restoring is somewhat more efficient, partly as a side effect of wider memory.

[1] In ConTEXt macros were always 'long' and never 'outer'. Most commands were unexpandable (also in MkII, pre ε-TEX). So, users won't notice this.

Language

Language control settings now use less parameters but instead use bit sets. Only basic parameters are stored in the format file now, and there are all kinds of other small improvements.

Typesetting

Attributes (the lists and states) are implemented more efficiently.

The paragraph state is stored with the paragraph. Paragraphs can be normalized and options are now set with bit sets. Directions are mostly gone; it's now up to the back-end.

Boxes carry orientation-related information (offsets, rotation, etc). Migrated content is optionally kept with boxes. Some nodes carry more information.

Math

Some math concepts have been extended like the prescripts and control over styles. There are plenty of new control details. The math parameter settings obey grouping in a math list.

We can have math in discretionaries in text and more advanced discretionaries in math as well.

Fonts

Font specification information no longer uses the string pool, which saves a lot. Of course we still have the basic font handler. We only store what is needed for traditional TEX font handling.

Virtual fonts are even more virtual (also a back-end thing), so we can have more features.

The code

Artifacts from Pascal and cweb have been removed.

Languages, fonts, marks etc. are no longer 'register'-based.

The token interface is more abstract and no longer presents strange numbers.

Some internals have been reconstructed because of the cleaner Lua interfacing. A side-effect of this is better abstraction of the equivalent ranges.

The code has been made more abstract (and looks easier in e. g. Visual Studio). Readability of the code is constantly improving (the usual: has to look okay in my editor).

The compile farm is used to check if compilation works out-of-the-box. Compilation is fast and easy, otherwise this project would not be possible.

The code has been made mostly independent of specific operating system needs. Wide characters are dealt with in the Windows interfaces.

Libraries

We really want to stay lean and mean: the engine is also a Lua engine.

All the required code is included in the distribution. There are a few libraries included but these are small, old and stable. In addition, some of helper libraries have been included such as Pawel's `pplib` library.

What we ship is what you get: ConTEXt will not depend on more than what we ship. If something is updated (at all), the output is checked for differences first.

The Lua engine

We use the latest (even alpha) Lua (5.4) because LuaMetaTEX is a good test. There is no support for LuaJIT, and the ffi interface is gone.

There is a limited set of libraries that we support, but no code is (or will be) included.

There are less callbacks because we only have a frontend.

There are more token scanners and some options have been added.

Efficiency

We benefit from wider memory words allowing some constructs to go. The core engine performs a bit better, but there isn't much to gain in that regard.

The format file is smaller and no longer compressed. Dumping the format has been made a bit more robust and faster.

There are more statistics (also as side effect of memory management).

Memory management is now mostly dynamic and utilization is much lower. The lot runs quite well on a Raspberry Pi 4, for example. We managed to keep the binary below 3 MB.

We want to be prepared for future architectures.

Upgraded MetaPost

All the old 8-bit font stuff has been stripped from the MetaPost library.

The library no longer has a PostScript backend.

It provides scanners that make extensions possible. There are a few additions to the MetaPost library, like pre/postscripts for clip and bounding boxes.

All file I/O goes via Lua.

Praise for the users

Much has been done, and I probably have forgotten to mention a lot.

The number of bugs is relative small compared to what gets changed and added. The test suite is run frequently to check for bugs and performance.

I could only do this because the ConTEXt users are exceedingly tolerant. Some seem to constantly check for updates which helps with faster testing.

The ConTEXt code base gets stepwise adapted (split files) which again forces users to test. It takes a lot of time because we take small steps in order not to mess up.

I would not do it without the positive attribute of the ConTEXt users. It's all about motivation and I thank the ConTEXt users for providing this friendly and non-competitive bubble!

Todo

- Maybe add some more sanity checks in order to catch errors intruded by callbacks. Maybe add some more tracing too.
- Explore variants, like having registers in dedicated eqtb tables so that we can allocate them dynamically (mostly for the fun of doing it).
- Add some more documentation (read: addition cq. remarks about where the original documentation no longer applies, but we have years for doing that).
- Update the manual (which is done occasionally in batch based on print-outs; there is no real need to hurry because we're still experimenting).
- Apply some of the new stuff in LMTX. Take up some challenges.
- Wrap up new functionality (once it's stable) in articles and other documents.

And LuaTEX?

Of course LuaTEX will be maintained! After all, MkIV needs it and it serves as reference for the front-end rendering and back-end generation when we're messing around with LuaMetaTEX.

It is used by LATEX and there are now also plain inspired packages. Because there are spin-offs (LATEX has settled on a version with built-in font processing) we cannot change much. And LuaTEX being nicely integrated into TEXLive is another argument for not touching it too much.

I have no clue of LuaTEX usage but that fact alone already makes an argument for being even more careful. It's a bad advertisement for TEX when users who use the low level interfaces get confronted with conceptual changes.

So in the end not much will be backported to LuaTEX: At some point the code base became too different and it's the price paid for the demanded stability. This way we cannot introduce new bugs and it doesn't pay off either.

But, a few non-intrusive changes might actually trickle into it in due time. Out of self-interest, it might help to share some code between MkIV and LMTX.