

LUAMETATEX

where do we stand

context 2020 meeting

When it started

- About three years ago the idea came up to go this route.
- At the 2018 meeting it was first mentioned and those present were okay with it.
- Early 2019 the first beta release took place.
- At the 2019 meeting the first more official version was presented.
- Around the 2020 meeting we have more or less arrived at what I had in mind.
- At the 2021 meeting I expect the code to be stable and 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 those changes makes me more daring.
- We aim to get less (but more efficient) macro code that on the average looks better.

Why it started

- There was an increasing pressure for a stable LuaT_EX.
- There should be no more changes to the interfaces, no more extensions.
- One can run into interesting comments on the web (as usual), like
 - The LuaT_EX program has ‘many bugs’.
 - The LuaT_EX manual is bad.
 - The LuaT_EX program is too slow to be useful.
 - The LuaT_EX program will never end up in distributions.
 - The LuaT_EX project is funded and developed in a commercial setting.
- I won’t comment on how I read these (demotivating) comments because . . .
- . . . it anyway often says more about the writer (attitudes) than about LuaT_EX.
- I also looks like (non ConT_EXt) users are charmed by LuaT_EX, and the more they code, the more we need to freeze.
- So, hopefully, the LuaMetaT_EX development does not interfere badly with developments outside the ConT_EXt community.

The development

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

- LuaTeX started out as cweb code . . . that eventually became just C . . . which in LuaMetaTeX has been detached from the (complex) infrastructure.
- The basic idea is to only keep the core of TeX, but for instance font loading, file handling and the backend are gone.
- As a consequence the code has been reorganized (shuffled around).
- I experimented a lot without bothering about usage elsewhere and I like the result so far.
- The ConTeXt distribution will at some point ship with the source.

File handling

- All file handling goes via Lua, also read and write related primitives.
- The same is true for terminal (console) handling.
- Part of that (the writing) was actually kind of extension code in T_EX and partly a system dependency.
- The ε -T_EX pseudo file `\scantokens` primitive uses the same mechanism as Lua does.

The macro machinery

- There are extensions to the way macro arguments are handled (less clumsy macros).
- There are extra if tests (makes for nicer macros).
- Else branches in conditions can be collapsed using `\orelse` and `\orunless` which gives cleaner low level code.
- Tracing gives more detail about node properties and also shows 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 a side effect of wider memory).

¹ In ConT_EXt macros were always `\long` and never `\outer`. Most commands were unexpandable (also in MkII, pre ϵ -T_EX). So, users won’t notice this.

Language

- Language control settings now use less parameters but bit sets instead.
- Only basic parameters are stored in the format file now.
- There are all kind of 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.
- Boxes carry orientation related information (offsets, rotation, etc).
- Some nodes carry more information.
- Directions are mostly gone (it's up to the backend).
- Migrated content is optionally kept with boxes.

Math

- Some math concepts have been extended (like prescripts and some more 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 $\text{T}_\text{E}\text{X}$ font handling.
- Virtual fonts are even more virtual (also a backend 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 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).
- The compile farm is used to check if compilation works out of the box.
- Compilation is fast and easy, otherwise this project was not possible.
- Readability of the code is constantly improved (the usual: has to look okay in my editor).
- The code has been made mostly independent of specific operating system needs.
- Wide characters are dealt with in Windows interfaces.

Libraries

- We really want to stay lean and mean: the engine is also a Lua engine.
- All code is included, a few libraries are used, but these are small, old and stable.
- In addition some helper libraries are made (including pplib by Pawel).
- What we ship is what you get: ConT_EXt will not depend on more than that.
- If something is updated (at all) the differences are checked 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 (and 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 some more from the wider memory words (some constructs could go).
- The format file is smaller and not longer compressed.
- Memory management is now mostly dynamic and usage is much lower.
- There are more statistics (also as side effect of memory management).
- Dumping the format has been made a bit more robust and is faster.
- The core engine performs a bit better (machines don't get that much faster).
- We want to be prepared for future architectures.
- We manage to keep the binary way below 3 MB.
- The lot runs quite well on e.g. a Raspberry Pi 4.

Upgraded MetaPost

- All (eight bit) font stuff has been stripped from the MetaPost library.
- The library no longer has a PostScript backend.
- The library provides scanners that make extensions possible.
- All file io goes via Lua.
- There are a few additions like pre/postscripts for clip and bounding boxes.

Praise for the users

- Much has been done and I probably forget to mention a lot.
- The number of bugs is relative small compared to what gets changed and added.
- The test suite gets ran very often, also to check if performance is okay.
- I could only do this because the ConT_EXt users are so tolerant.
- Some seem to constantly check for updates so they help with fast testing.
- The ConT_EXt 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 ConT_EXt users.
- It's all about motivation and I thank the ConT_EXt 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 still experiment).
- 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 with LuaMetaTeX.
- It is used by L^ATeX and there are now also plain inspired packages. Because there are spin-offs (L^ATeX has settled on a version with built-in font processing) we cannot change much.
- And LuaTeX being nicely integrated into T_EXLive 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 bad advertisement for T_EX when users who use the low level interfaces get confronted with conceptual changes.
- So in the end not much will be back ported to LuaTeX: at some point the code base became too different and it's the price paid for the stability demand. That way we cannot introduce new bugs either. It also doesn't pay of.
- But, a few non-intrusive things might actually trickle into it in due time, also out of self interest: it might help to share code between MkIV and LMTX.