The Asian Journal of TEX, Volume 3, No. 1, June 2009
Article revision 2009/4/16

Halfway, the LuaTEX Project

Hans Hagen
PRAGMA ADE pragma@wxs.nl

Taco Hoekwater
Elvenkind B.V. taco@elvenkind.com

KEYWORDS LuaTEX, ConTEXt, OpenType, METAPOST, MetaTEX

ABSTRACT In this note we describe the current status of the LuaTEX project, including several topics, design principles, Lua scripting, I/O, interface, fonts, tokens, nodes, attributes, hyphenation, images, paragraph building, METAPOST, mathematics, page building, CWEB, cleanup, alignments, error handling, backend, ConTEXt MkIV, and the future.

Introduction

We are about halfway the LuaTEX project now. At the time of writing this document we are only a few days away from version 0.40 (the BachoTEX cq. TEX Live version) and around EuroTEX 2009 we will release version 0.50. Starting with version 0.30 (which we released around '2009 Conference and Annual Meeting of the Korean TEX Society') all one-decimal releases are supported and usable for (controlled) production work. We have always stated that all interfaces may change until they are documented to be stable, and we expect to document the first stable parts in version 0.50. Currently we plan to release version 1.00 sometime in 2012, 30 years after TEX82, with 0.60 and 0.70 in 2010, 0.80 and 0.90 in 2011. But of course it might turn out different.

In this update we assume that the reader knows what LuaTEX is and what it does.

Design principles

We started this project because we wanted an extensible engine and have chosen Lua as glue language. We do not regret this choice as it permitted us to open up TEX's internals pretty well. There have been a few extensions to TEX itself and there will be a few more but none of them are fundamental in the sense that they influence typesetting. Extending TEX in that area is up to the macro package writer who can use the Lua language combined with TEX macros. In a similar fashion we made some decisions about Lua libraries that are included. What we have now is what you will get. Future versions of LuaTEX will have the ability to load additional libraries but these will not be part of the core distribution. There is simply too much choice and we do not want to enter endless discussions about what is best. More flexibility would also add a burden on maintenance that we do not want. Portability has always been a virtue of TEX and we want to keep it that way.

Lua scripting

Before 0.40 there could be multiple instances of the Lua interpreter active at the same time, but we decided to limit the number of instances to just one. The reason is simple: sharing all functionality among multiple Lua interpreter instances does more bad than good and Lua has enough possibilities to create namespaces anyway. The new limit also simplifies the internal source code, which is a good thing. While the `\directlua` command is now sort of frozen, we might extend the functionality of `\latelua` especially in relation to what is possible in the backend. Both commands still accept a number but this now refers to an index in a user-definable name table that will be shown when an error occurs.

Input and output

The current LuaTEX release permits multiple instances of kpse which can be handy if you mix for instance a macro package and mplib, as both have there own `progname` (and engine) namespace. However, right from the start it has been possible to bring most input under Lua control and one can overload the usual kpse mechanisms. This is what we do in ConTEXt (and probably only there).

Logging etc. is also under Lua control. There is no support for writing to TEX's opened output channels except for the log and the terminal. We are investigating limited write control to numbered channels but this has a very low priority.

Reading from zip files and sockets has been available for a while now.

Among the first things that has been implemented is a mechanism for managing category codes (`\catcode`) although this is not really needed for practical usage as we aim at full compatibility. It just makes printing back to TEX from Lua a bit more comfortable.

Interface to TEX

Registers can always be accessed from Lua by number and (when defined at the TEX end) also by name. When writing to a register grouping is honored. Most internal registers can be accessed (mostly read-only). Box registers can be manipulated but users need to be aware of potential memory management issues.

There will be provisions to use the primitives related to setting codes (lowercase codes and such). Some of this functionality will be available in version 0.50.

Fonts

The internal font model has been extended to the full Unicode range. There are readers for OpenType, Type1, and traditional TEX fonts. Users can create virtual fonts on the fly and have complete control over what goes into TEX. Font specific features can either be mapped onto the traditional ligature and kerning mechanisms or be implemented in Lua.

We use code from FontForge that has been stripped to get a smaller code base. Using the FontForge code has the advantage that we get a similar view on the fonts in LuaTEX as in this editor which makes debugging easier and developing fonts more convenient.

The interface is already rather stable but some of the keys in loaded tables might change. Almost all of the font interface will be stable in version 0.50.

Tokens

It is possible to intercept tokenization. Once intercepted, a token table can be manipulated before being piped back into LuaTEX. We still support Omega's translation processors but that might become obsolete at some point.

Future versions of LuaTEX might use Lua's so called user data concept but the interface will mostly be the same. Therefore this subsystem will not be frozen yet in version 0.50.

Nodes

Users have access to the node lists in various stages. This interface has already been quite stable for some time but some cleanup might still take place. Currently the node memory maintenance is still explicit, but we will eventually make releasing unused nodes automatic.

We have plans for keeping more extensive information within a paragraph (initial whatsit) so that one can build alternative paragraph builders in Lua. There will be a vertical packer (in addition to the horizontal packer) and we will open up the page builder (inserts etc.). The basic interface will be stable in version 0.50.

Attributes

This new kid on the block is now available for most subsystems but we might change some of its default behavior. As of 0.40 you can also use negative values for attributes. The original idea of using negative values for special purposes has been abandoned as we consider a secondary (faster and more efficient) limited variant. The basic principles will be stable around version 0.50, but we reserve the freedom to change some aspects of attributes until we reach version 1.00.

Hyphenation

In LuaTEX we have clearly separated hyphenation, ligature building and kerning. Managing patterns as well as hyphenation is reimplemented from scratch but uses the same principles as traditional TEX. Patterns can be loaded at run time and exceptions are quite efficient now. There are a few extensions, like embedded discretionaries in exceptions and pre- as well as posthyphens.

On the agenda is fixing some `hyphenchar` related issues and future releases might deal with compound words as well. There are some known limitations that we hope to have solved in version 0.50.

Images

Image handling is part of the backend. This part of the pdfTEX code has been rewritten and can now be controlled from Lua. There are already a few more options than in pdfTEX (simple transformations). The image code will also be integrated in the virtual font handler.

Paragraph building

The paragraph builder has been rewritten in C (soon to be converted back to CWEB). There is a callback related to the builder so it is possible to overload the default line breaker by one written in Lua.

There are no further short-term revisions on the agenda, apart from writing an advanced (third order) Arabic routine for the Oriental TEX project.

Future releases may provide a bit more control over `\parshapes` and multiple paragraph shapes.

METAPOST

The closely related mplib project has resulted in a METAPOST library that is included in LuaTEX. There can be multiple instances active at the same time and METAPOST processing is very fast. Conversion to pdf is to be done with Lua.

On the todo list is a bit more interoperability (pre- and postscript tables) and this will make it into release 0.50 (maybe even in version 0.40 already).

Mathematics

Version 0.50 will have a stable version of Unicode math support. Math is backward compatible but provides solutions for dealing with OpenType math fonts. We provide math lists in their intermediate form (noads) so that it is possible to manipulate math in great detail.

The relevant math parameters are reorganized according to what OpenType math provides (we use Cambria as reference). Parameters are grouped by style. Future versions of LuaTEX will build upon this base to provide a simple mechanism for switching style sets and font families in-formula.

There are new primitives for placing accents (top and bottom variants and extensible characters), creating radicals, and making delimiters. Math characters are permitted in text mode.

There will be an additional alignment mechanism analogous to what MathML provides. Expect more.

Page building

Not much work has been done on opening up the page builder although we do have access to the intermediate lists. This is unlikely to happen before 0.50.

Going CWEB

After releasing version 0.50 around EuroTEX 2009 there will be a period of relative silence. Apart from bug fixes and (private) experiments there will be no release for a while. At the time of the 0.50 release the LuaTEX source code will probably be in plain C completely. After that is done, we will concentrate hard on consolidating and upgrading the code base back into CWEB.

Cleanup

Cleanup of code is a continuous process. Cleanup is needed because we deal with a merge of traditional TEX, ε-TEX extensions, pdfTEX functionality and some Omega (Aleph) code.

Compatibility is a prerequisite, with the exception of logging and rather special ligature reconstruction code.

We also use the opportunity to slowly move away from all the global variables that are used in the Pascal version.

Alignments

We do have some ideas about opening up alignments, but it has a low priority and it will not happen before the 0.50 release.

Error handling

Once all code is converted to CWEB, we will look into error handling and recovery. It has no high priority as it is easier to deal with after the conversion to CWEB.

Backend

The backend code will be rewritten stepwise. The image related code has already been redone, and currently everything related to positioning and directions is redesigned and made more consistent. Some bugs in the Aleph code (inherited from Omega) have been removed and we are trying to come up with a consistent way of dealing with directions. Conceptually this is somewhat messy because much directionality is delegated to the backend.

We are experimenting with positioning (preroll) and better literal injection. Currently we still use the somewhat fuzzy pdfTEX methods that evolved over time (direct, page and normal injection) but we will come up with more clear model.

Accuracy of the output (pdf) will be improved and character extension (hz) will be done more efficient. Experimental code seems to work okay. This will become available from release 0.40 and onwards and further cleanup will take place when the CWEB code is there as much of the pdf backend code is already C.

ConTEXt MkIV

When we started with LuaTEX we decided to use a branch of ConTEXt for testing as it involves quite drastic changes, many rewrites, a tight connection with binary versions, etc.

As a result for some time we now have two versions of ConTEXt: MkII and MkIV, where the first one targets at pdfTEX and XƎTEX, and the second one is exclusively using LuaTEX. Although the user interface is downward compatible the code base starts to diverge more and more. Therefore at the last ConTEXt meeting it was decided to freeze the current version of MkII and only apply bug fixes and an occasional simple extension.

This policy change opened the road to rather drastic splitting of the code, also because full compatibility between MkII and MkIV is not required. Around LuaTEX version 0.40 the new, currently still experimental, document structure related code will be merged into the regular MkIV version. This might have some impact as it opens up new possibilities.

The future

In the future, MkIV will try to create (more) clearly separated layers of functionality so that it will become possible to make subsets of ConTEXt for special purposes. This is done under the name MetaTEX. Think of layering like:

- io, catcodes, callback management, helpers
- input regimes, characters, filtering
- nodes, attributes and noads
- user interface
- languages, scripts, fonts and math
- spacing, par building and page construction
- XML, graphics, METAPOST, job management, structure (huge impact)
- modules, styles, specific features
- tools

OpenType and CJK fonts

At this moment MkIV is already quite capable of dealing with OpenType fonts. The driving force behind this is the Oriental TEX project which brings along some very complex and feature rich Arabic font technology. Much time has gone into reverse engineering the specification and behaviour of how these fonts behave in Uniscribe (which we use as reference for Arabic).

Dealing with the huge CJK fonts is less a font issue and more a matter of node list processing. Around '2009 Conference and Annual Meeting of the Korean TEX Society' we got much of the machinery working, thanks to discussions on the spot and on the mailing list.

Math machinery

Between LuaTEX versions 0.30 and 0.40 the math machinery was opened up (stage one). In order to test this new functionality, MkIV's math subsystem (that was then already partially Unicode aware) had to be adapted.

First of all Unicode permits us to use only one math family and so MkIV now does that. The implementation uses Microsoft's Cambria Math font as a benchmark. It creates virtual fonts from the other (old and new) math fonts so they appear to match up to Cambria Math. Because the TEX Gyre math project is not yet up to speed MkIV currently uses virtual variants of these fonts that are created at run time. The missing pieces in for instance Latin Modern and friends are compensated for by means of virtual characters.

Because it is now possible to parse the intermediate noad lists MkIV can do some manipulations before the formula is typeset. This is for instance used for alphabet remapping, forcing sizes, and spacing around punctuation.

Although MkIV already supports most of the math that users expect there is still room for improvement once there is even more control over the machinery. This is possible because MkIV is not bound to downward compatibility.

As with all other LuaTEX related MkIV code, it is expected that we will have to rewrite most of the current code a few times as we proceed, so MkIV math support is not yet stable either. We can take such drastic measures because MkIV is still experimental and because users are willing to do frequent synchronous updating of macros and engine. In the process we hope to get away from all ad-hoc boxing and kerning and whatever solutions for creating constructs, by using the new accent, delimiter, and radical primitives.

Tracing and testing

Whenever possible we add tracing and visualization features to ConTEXt because the progress reports and articles need them. Recent extensions concerned tracing math and tracing OpenType processing.

The OpenType tracing options are a great help in stepwise reaching the goals of the Oriental TEX project. This project gave the LuaTEX project its initial boost and aims at high quality right to left typesetting. In the process complex (test) fonts are made which, combined with the tracing mentioned helps us to reveal the secrets of OpenType.