where 1

does 2

TEX 3

end, 4

Lua 5

start 6

and 7

vise 8

versa 9

We see no reason to fundamentally change the concept of TEX as it looks like most users are happy with it. We don't implement new solutions as it is pretty hard to come up with a common view on how pending issues can be solved.

1

In ConTEXt, we started with just calling Lua and piping something into TEX (e.g. calcmath). At that stage Lua was completely separated from TEX, there was only a caller primitive and a print function. This already permitted interesting expandable testing macro that were previously impossible.

3

Major chunks of Lua code started showing up when the font loading was opened up. So next we brought loading under Lua control which then includes preparing for OpenType processing.

9

Input regimes (including utf-16) was reimplemented in Lua, and all old code was kicked out of the ConTEXt kernel. This was the first code that was replaced.

6

A bit more ambitious effort was replacing all file handling in Lua, including all things normally done by the kpse library. Reading from zip files, http, ftp etc. has become possible. TEX itself is rather unaware of this. About at the same time we moved much multipass data to Lua.

5

We open up the internals of TEX but try to remain compatible where possible. The rest is up to the macro programmer and user. We will provide tools to improve existing and implement new solutions. This is up to the macro package and here we will focus on how ConTEXt MkIV does is.

2

When we got access to node lists, it became possible to manipulate for instance glyph related data. Suddenly we could provide more robust solutions for case swapping and such.

8

When we got access to registers, we started playing with calculating properties in Lua, but this was not that spectacular. Only counters and dimensions (including box dimensions) could be accessed.

4

More complex Lua scripting was added and again existing TEX code was replaced, like MetaPost conversion which involved piping data to Lua as well as print back pdf literals. Verbatim (pretty printing) and buffers are now mostly dealt with by Lua as well.

7

Reading from afm files replaced tfm when possible . . . we have wide Type1 fonts now. As a result font encoding has been removed from MkIV. Only math still needs tfm files but that will go too.

1

Runtime virtual font building can be used to construct missing glyphs. Instead of frozen features we can support them dynamically when needed. Things like this are kind of new to TEX but are no real extensions, they are just made possible by opening up.

3

Currently the sectioning mechanisms, numbering and lists are reimplemented in Lua, apart from the typesetting part. More data can be carries around and more status information is kept.

8

Driven by the Oriental TEX project we started writing support for advanced OpenType features. Everything is completely under Lua control. Surprisingly the rather major node-crunching is quite doable because Lua is so fast.

2

We started experimenting with advanced vertical spacing models by manipulating node lists but we're a bit ahead of what is opened up now. The next stop is opening up math. In MkIV we already have started reorganizing math (needed for projects). We really need the TEX-Gyre math fonts!

5

The stream driven MkII xmlhandling was replaced by a tree based method that permits arbitrary access, flushing and manipulation of the tree. As a usage case MathML support is reimplemented.

6

Bits and pieces of the typesetting options are replaced and extended and more is to come. Eventually we will let Lua do what it can do best, and let TEX deal with the typesetting.

9

The new img library permitted a reimplementation of graphic inclusion and its components: figure libraries, fallbacks, conversion, tracing, etc.

7

The generic attribute mechanism (each node can have one or more attributes) triggered a rewrite of color support. This replaced much code, was not much faster, but more robust (due to less interference, i.e. no whatsits). It needs a cleanup with respect to the backend.

4

You can use Lua to replace typesetting components of TEX, like hyphenation, kerning, paragraph building.

8

You can pass data to Lua, manipulate it, and feed back (maybe something) completely different to TEX.

3

You can replace components of TEX, like file handling, by Lua code which is more flexible.

4

You can set attributes at the TEX end and at various moments decide to use their values to manipulate node lists.

7

You can get information from TEX (registers), do some calculations, and feed back something to TEX.

2

You can use Lua to implement complex input and font manipulations for instance bidirectional typesetting and OpenType features.

6

You can enhance TEX with new features, not by hardcoding it in the core engine but by using (macro specific) Lua code.

9

You can replace more fundamental parts of TEX, like font loading and definitions, and for instance create virtual fonts.

5

You can use just Lua, forget about the TEX internals part and just print things to TEX.

1