The luafication of TEX and ConTEXt

Introduction

Here I will present the current stage of LuaTEX around beta stage 2, and discuss the impact so far on ConTEXt MkIV that we use as our testbed. I'm writing this at the end of February 2008 as part of the series of regular updates on LuaTEX. As such, this report is part of our more or less standard test document (`mk.tex`). More technical details can be found in the reference manual that comes with LuaTEX. More information on MkIV is available in the ConTEXt mailing lists, Wiki, and `mk.pdf`.

For those who never heard of LuaTEX: this is a new variant of TEX where several long pending wishes are fulfilled:

- combine the best of all TEX engines
- add scripting capabilities
- open up the internals to the scripting engine
- enhance font support to OpenType
- move on to Unicode
- integrate MetaPost

There are a few more wishes, like converting the code base to c but these are long term goals.

The project started a few years ago and is conducted by Taco Hoekwater (Pascal and c coding, code base management, reference manual), Hartmut Henkel (pdf backend, experimental features) and Hans Hagen (general overview, Lua and TEX coding, website). The code development got a boost by a grant of the Oriental TEX project (project lead: Idris Samawi Hamid) and funding via the tug. The related MPlib project by the same team is also sponsored by several user groups. The very much needed OpenType fonts are also a user group funded effort: the Latin Modern and TEX Gyre projects (project leads: Jerzy Ludwichowski, Volker RW Schaa and Hans Hagen), with development (the real work) by: Bogusław Jackowski and Janusz Nowacki.

One of our leading principles is that we focus on opening up. This means that we don't implement solutions (which also saves us many unpleasant and everlasting discussions). Implementing solutions is up to the user, or more precisely: the macro package writer, and since there are many solutions possible, each can do it his or her way. In that sense we follow the footsteps of Don Knuth: we make an extensible tool, you are free to like it or not, you can take it and extend it where needed, and there is no need to bother us (unless of course you find bugs or weird side effects). So far this has worked out quite well and we're confident that we can keep our schedule.

We do our tests of a variant of ConTEXt tagged MkIV, especially meant for LuaTEX, but LuaTEX itself is in no way limited to or tuned for ConTEXt. Large chunks of the code written for MkIV are rather generic and may eventually be packaged as a base system (especially font handling) so that one can use LuaTEX in rather plain mode. To a large extent MkIV will be functionally compatible with MkII, the version meant for traditional TEX, although it knows how to profit from XETEX. Of course the expectation is that certain things can be done better in MkIV than in MkII.

Status

By the end of 2007 the second major beta release of LuaTEX was published. In the first quarter of 2008 Taco would concentrate on MPlib, Hartmut would come up with the first version of the image library while I could continue working on MkIV and start using LuaTEX in real projects. Of course there is some risk involved in that, but since we have a rather close loop for critical bug fixes, and because I know how to avoid some dark corners, the risk was worth taking.

What did we accomplish so far? I can best describe this in relation to how ConTEXt MkIV evolved and will evolve. Before we do this, it makes sense to spend some words on why we started working on MkIV in the first place.

When the LuaTEX project started, ConTEXt was about 10 years in the field. I can safely say that we were still surprised by the fact that what at first sight seems unsolvable in TEX somehow could always be dealt with. However, some of the solutions were rather tricky. The code evolved towards a more or less stable state, but sometimes depended on controlled processing. Take for instance backgrounds that can span pages and columns, can be nested and can have arbitrary shapes. This feature has been present in ConTEXt for quite a while, but it involves an interplay between TEX and MetaPost. It depends on information collected in a previous run as well as processing of graphics.

This means that by now ConTEXt is not just a bunch of TEX macros, but also closely related to MetaPost. It also means that processing itself is by now rather controlled by a wrapper, in the case of MkII called TEXexec. It may sound complicated, but the fact that we have implemented workflows that run unattended for many years and involve pretty complex layouts and graphic manipulations demonstrates that in practice it's not as bad as it may sound.

With the arrival of LuaTEX we not only have a rigourously updated TEX engine, but also get MetaPost integrated. Even better, the scripting language Lua is not only used for opening up TEX, but is also used for all kind of management tasks. As a result, the development of MkIV not only concerns rewriting whole chunks of ConTEXt, but also results in a set of new utilities and a rewrite of existing ones. Since dealing with MkIV will demand some changes in the way users deal with ConTEXt I will discuss some of them first. It also demonstrates that LuaTEX is more than just TEX.

Utilities

There are two main scripts: luatools and mtxrun. The first one started as a replacement for kpsewhich but evolved into a base tool for generating (tds) file databases and generating formats. In MkIV we replace the regular file searching, and therefore we use a different database model. That's the easy part. More tricky is that we need to bootstrap MkIV into this alternative mode and when doing so we don't want to use the `kpse` library because that would trigger loading of its databases. To discuss the gory details here might cause users to refrain from using LuaTEX so we stick to a general description.

- When generating a format, we also generate a bootstrap Lua file. This file is compiled to bytecode and is put alongside the format file. The libraries of this bootstrap file are also embedded in the format.
- When we process a document, we instruct LuaTEX to load this bootstrap file before loading the format. After the format is loaded, we re-initialize the embedded libraries. This is needed because at that point more information may be available than at loading time. For instance, some functionality is available only after the format is loaded and LuaTEX enters the TEX state.
- File databases, formats, bootstrap files, and runtime-generated cached data is kept in a tds tree specific cache directory. For instance, OpenType font tables are stored on disk so that next time loading them is faster.

Starting LuaTEX and MkIV is done by luatools. This tool is generic enough to handle other formats as well, like mptopdf or Plain. When you run this script without argument, you will see:

```
version 1.1.1 - 2006+ - PRAGMA ADE / CONTEXT

--generate         generate file database
--variables        show configuration variables
--expansions       show expanded variables
--configurations   show configuration order
--expand-braces    expand complex variable
--expand-path      expand variable (resolve
                                        paths)
--expand-var       expand variable (resolve
                                    references)
--show-path        show path expansion of ...
--var-value        report value of variable
--find-file        report file location
--find-path        report path of file
--make or --ini    make luatex format
--run or --fmt=    run luatex format
--luafile=str      lua inifile (default is
                                <progname>.lua)
--lualibs=list     libraries to assemble
                                     (optional)
--compile          assemble and compile lua
                                       inifile
--verbose          give a bit more info
--minimize         optimize lists for format
--all              show all found files
--sort             sort cached data
--engine=str       target engine
--progname=str     format or backend
--pattern=str      filter variables
--lsr              use lsr and cnf directly
```

For the Lua based file searching, luatools can be seen as a replacement for mktexlsr and kpsewhich and as such it also recognizes some of the kpsewhich flags. The script is self contained in the sense that all needed libraries are embedded. As a result no library paths need to be set and packaged. Of course the script has to be run using LuaTEX itself. The following commands generate the file databases, generate a ConTEXt MkIV format, and process a file:

```
luatools --generate
luatools --make --compile cont-en
luatools --fmt=cont-en somefile.tex
```

There is no need to install Lua in order to run this script. This is because LuaTEX can act as such with the advantage that the built-in libraries are available

```
version 1.0.2 - 2007+ - PRAGMA ADE / CONTEXT

--script              run an mtx script
--execute             run a script or program
--resolve             resolve prefixed arguments
--ctxlua              run internally (using preloaded libs)
--locate              locate given filename

--autotree            use texmf tree cf.\ environment settings
--tree=pathtotree     use given texmf tree (def: 'setuptex.tmf')
--environment=name    use given (tmf) environment file
--path=runpath        go to given path before execution
--ifchanged=filename  only execute when given file has changed
--iftouched=old,new   only execute when given file has changed

--make                create stubs for (context related) scripts
--remove              remove stubs (context related) scripts
--stubpath=binpath    paths where stubs wil be written
--windows             create windows (mswin) stubs
--unix                create unix (linux) stubs

--verbose             give a bit more info
--engine=str          target engine
--progname=str        format or backend

--edit                launch editor with found file
--launch (--all)      launch files (assume os support)

--intern              run script using built-in libraries
```

Figure 1. mtxrun help information

too, for instance the Lua file system `lfs`, the zip file manager `zip`, the Unicode libary `unicode`, `md5`, and of course some of our own.

luatex	a Lua–enhanced TEX engine
texlua	a Lua engine enhanced with some libraries
texluac	a Lua bytecode compiler enhanced with some libraries

In principle `luatex` can perform all tasks but because we need to be downward compatible with respect to the command line and because we want Lua compatible variants, you can copy or symlink the two extra variants to the main binary.

The second script, mtxrun, can be seen as a replacement for the Ruby script texmfstart, a utility whose main task is to launch scripts (or documents or whatever) in a tds tree. The mtxrun script makes it possible to get away from installing Ruby and as a result a regular TEX installation can be made independent of scripting tools.

The help information is shown in figure 1. It gives an impression of what the script does: running other scripts, either within a certain tds tree or not, and either conditionally or not. Users of ConTEXt will probably recognize most of the flags. As with texmfstart, arguments with prefixes like `file:` will be resolved before being passed to the child process.

The first option, `--script` is the most important one and is used like:

```
mtxrun --script fonts --reload
mtxrun --script fonts --pattern=lm
```

In MkIV you can access fonts by filename or by font name, and because we provide several names per font you can use this command to see what is possible. Patterns can be Lua expressions, as demonstrated in figure 2.

```
mtxrun --script font  --list --pattern=lmtype.*regular

lmtypewriter10-capsregular   LMTypewriter10-CapsRegular   lmtypewriter10-capsregular.otf
lmtypewriter10-regular       LMTypewriter10-Regular       lmtypewriter10-regular.otf
lmtypewriter12-regular       LMTypewriter12-Regular       lmtypewriter12-regular.otf
lmtypewriter8-regular        LMTypewriter8-Regular        lmtypewriter8-regular.otf
lmtypewriter9-regular        LMTypewriter9-Regular        lmtypewriter9-regular.otf
lmtypewritervarwd10-regular  LMTypewriterVarWd10-Regular  lmtypewritervarwd10-regular.otf
```

Figure 2. Example of a `mtxrun --script font` run.

A simple

```
mtxrun --script fonts
```

gives:

```
version 1.0.2 - 2007+ - PRAGMA ADE / CONTEXT
                                    | font tools

--reload       generate new font database
--list         list installed fonts
--save         save open type font in raw table

--pattern=str  filter files
--all          provide alternatives
```

In MkIV font names can be prefixed by `file:` or `name:` and when they are resolved, several attempts are made, for instance non-characters are ignored. The `--all` flag shows more variants.

Another example is:

```
mtxrun --script context --ctx=somesetup
                                    somefile.tex
```

Again, users of TEXexec may recognize part of this and indeed this is its replacement. Instead of TEXexec we use a script named `mtx-context.lua`. Currently we have the following scripts and more will follow:

The `babel` script is made in cooperation with Thomas Schmitz and can be used to convert babelized Greek files into proper utf. More of such conversions may follow. With `cache` you can inspect the content of the MkIV cache and do some cleanup. The `chars` script is used to construct some tables that we need in the process of development. As its name says, `check` is a script that does some checks, and in particular it tries to figure out if TEX files are correct. The already mentioned `context` script is the MkIV replacement of TEXexec, and takes care of multiple runs, preloading project specific files, etc. The `convert` script will replace the Ruby script `pstopdf`.

A rather important script is the already mentioned `fonts`. Use this one for generating font name databases (which then permits a more liberal access to fonts) or identifying installed fonts. The `unzip` script indeed unzips archives. The `update` script is still somewhat experimental and is one of the building blocks of the ConTEXt minimal installer system by Mojca Miklavec and Arthur Reutenauer. This update script synchronizes a local tree with a repository and keeps an installation as small as possible, which for instance means: no OpenType fonts for pdfTEX, and no redundant Type1 fonts for LuaTEX and XETEX.

The (for the moment) last two scripts are `watch` and `web`. We use them in (either automated or not) remote publishing workflows. They evolved out of the eXaMpLe framework which is currently being reimplemented in Lua.

As you can see, the LuaTEX project and its ConTEXt companion MkIV project not only deal with TEX itself but also facilitates managing the workflows. And the next list is just a start.

context	controls processing of files by MkIV
babel	conversion tools for LaTEX files
cache	utilities for managing the cache
chars	utilities used for MkIV development
check	TEX syntax checker
convert	helper for some basic graphic conversion
fonts	utilities for managing font databases
update	tool for installing minimal ConTEXt trees
watch	hot folder processing tool
web	utilities related to automate workflows

There will be more scripts. These scripts are normally rather small because they hook into mtxrun which provides the libraries. Of course existing tools remain part of the toolkit. Take for instance ctxtools, a Ruby script that converts font encoded pattern files to generic utf encoded files.

Those who have followed the development of ConTEXt will notice that we moved from utilities written in Modula to tools written in Perl. These were

later replaced by Ruby scripts and eventually most of them will be rewritten in Lua.

Macros

I will not repeat what is said already in the MkIV related documents, but stick to a summary of what the impact on ConTEXt is and will be. From this you can deduce what the possible influence on other macro packages can be.

Opening up TEX started with rewriting all io related activities. Because we wanted to be able to read from zip files, the web and more, we moved away from the traditional kpse based file handling. Instead MkIV uses an extensible variant written in Lua. Because we need to be downward compatible, the code is somewhat messy, but it does the job, and pretty quickly and efficiently too. Some alternative input media are implemented and many more can be added. In the beginning I permitted several ways to specify a resource but recently a more restrictive url syntax was imposed. Of course the file locating mechanisms provide the same control as provided by the file readers in MkII.

An example of reading from a zip file is:

```
\input zip:///archive.zip?name=blabla.tex
\input zip:///archive.zip?name=/path/blabla.tex
```

In addition one can register files, like:

```
\usezipfile[archive.zip]
\usezipfile[tex.zip][texmf-local]
\usezipfile[tex.zip?tree=texmf-local]
```

The last two variants register a zip file in the tds structure where more specific lookup rules apply. The files in a registered file are known to the file searching mechanism so one can give specifications like the following:

```
\input */blabla.tex
\input */somepath/blabla.tex
```

In a similar fashion one can use the `http`, `ftp` and other protocols. For this we use independent fetchers that cache data in the MkIV cache. Of course, in more structured projects, one will seldom use the `\input` command but use a project structure instead.

Handling of files rather quickly reached a stable state, and we seldom need to visit the code for fixes. Already after a few years of developing the first code for LuaTEX we reached a state of 'Hm, when did I write this?'. When we have reached a stable state I foresee that much of the older code will need a cleanup.

Related to reading files is the sometimes messy area of input regimes (file encoding) and font encoding, which itself relates to dealing with languages. Since LuaTEX is utf-8 based, we need to deal with file encoding issues in the frontend, and this is what Lua based file handling does. In practice users of LuaTEX will swiftly switch to utf anyway but we provide regime control for historic reasons. This time the recoding tables are Lua based and as a result MkIV has no regime files. In a similar fashion font encoding is gone: there is still some old code that deals with default fallback characters, but most of the files are gone. The same will be true for math encoding. All information is now stored in a character table which is the central point in many subsystems now.

It is interesting to notice that until now users have never asked for support with regards to input encoding. We can safely assume that they just switched to utf and recoded older documents. It is good to know that LuaTEX is mostly pdfTEX but also incorporates some features of Omega. The main reason for this is that the Oriental TEX project needed bidirectional typesetting and there was a preference for this implementation over the one provided by ε-TEX. As a side effect input translation is also present, but since no one seems to use it, that may as well go away. In MkIV we refrain from input processing as much as possible and focus on processing the node lists. That way there is no interference between user data, macro expansion and whatever may lead to the final data that ends up in the to-be-typeset stream. As said, users seem to be happy to use utf as input, and so there is hardly any need for manipulations.

Related to processing input is verbatim: a feature that is always somewhat complicated by the fact that one wants to typeset a manual about TEX in TEX and therefore needs flexible escapes from illustrative as well as real TEX code. In MkIV verbatim as well as all buffering of data is dealt with in Lua. It took a while to figure out how LuaTEX should deal with the concept of a line ending, but we got there. Right from the start we made sure that LuaTEX could deal with collections of catcode settings (those magic states that characters can have). This means that one has complete control at both the TEX and Lua end over the way characters are dealt with.

In MkIV we also have some pretty printing features, but many languages are still missing. Cleaning up the premature verbatim code and extending pretty printing is on the agenda for the end of 2008.

Languages also are handled differently. A major change is that pattern files are no longer preloaded but read in at runtime. There is still some relation between fonts and languages, no longer in the encoding but in dealing with OpenType features. Later we will do a

more drastic overhaul (with multiple name schemes and such). There are a few experimental features, like spell checking.

Because we have been using utf encoded hyphenation patterns for quite some time now, and because ConTEXt ships with its own files, this transition probably went unnoticed, apart maybe from a faster format generation and less startup time.

Most of these features started out as an experiment and provided a convenient way to test the LuaTEX extensions. In MkIV we go quite far in replacing TEX code by Lua, and how far one goes is a matter of taste and ambition. An example of a recent replacement is graphic inclusion. This is one of the oldest mechanisms in ConTEXt and it has been extended many times, for instance by plugins that deal with figure databases (selective filtering from pdf files made for this purpose), efficient runtime conversion, color conversion, downsampling and product dependent alternatives.

One can question if a properly working mechanism should be replaced. Not only is there hardly any speed to gain (after all, not that many graphics are included in documents), a Lua–TEX mix may even look more complex. However, when an opened-up TEX keeps evolving at the current pace, this last argument becomes invalid because we can no longer give that TEXie code to Lua. Also, because most of the graphic inclusion code deals with locating files and figuring out the best quality variant, we can benefit much from Lua: file handling is more robust, the code looks cleaner, complex searches are faster, and eventually we can provide way more clever lookup schemes. So, after all, switching to Lua here makes sense. A nice side effect is that some of the mentioned plugins now take a few lines of extra code instead of many lines of TEX. At the time of writing this, the beta version of MkIV has Lua based graphic inclusion.

A disputable area for Luafication is multipass data. Most of that has already been moved to Lua files instead of TEX files, and the rest will follow: only tables of contents still use a TEX auxiliary file. Because at some point we will reimplement the whole section numbering and cross referencing, we postponed that till later. The move is disputable because in the end, most data ends up in TEX again, which involves some conversion. However, in Lua we can store and manipulate information much more easily and so we decided to follow that route. As a start, index information is now kept in Lua tables, sorted on demand, depending on language needs and such. Positional information used to take up much hash space which could deplete the memory pool, but now we can have millions of tracking points at hardly any cost.

Because it is a quite independent task, we could rewrite the MetaPost conversion code in Lua quite early in the development. We got smaller and cleaner code, more flexibility, and also gained some speed. The code involved in this may change as soon as we start experimenting with MPlib. Our expectations are high because in a bit more modern designs a graphic engine cannot be missed. For instance, in educational material, backgrounds and special shapes are all over the place, and we're talking about many MetaPost runs then. We expect to bring down the processing time of such documents considerably, if only because the MetaPost runtime will be close to zero (as experiments have shown us).

While writing the code involved in the MetaPost conversion a new feature showed up in Lua: `lpeg`, a parsing library. From that moment on `lpeg` was being used all over the place, most noticeably in the code that deals with processing xml. Right from the start I had the feeling that Lua could provide a more convenient way to deal with this input format. Some experiments with rewriting the MkII mechanisms did not show the expected speedup and were abandoned quickly.

Challenged by `lpeg` I then wrote a parser and started playing with a mixture of a tree based and stream approach to xml (MkII is mostly stream based). Not only is loading xml code extremely fast (we used 40 megaByte files for testing), dealing with the tree is also convenient. The additional MkIV methods are currently being tested in real projects and so far they result in an acceptable and pleasant mix of TEX and xml. For instance, we can now selectively process parts of the tree using path expressions, hook in code, manipulate data, etc.

The biggest impact of LuaTEX on the ConTEXt code base is not the previously mentioned mechanisms but one not yet mentioned: fonts. Contrary to XETEX, which uses third party libraries, LuaTEX does not implement dealing with font specific issues at all. It can load several font formats and accepts font data in a well-defined table format. It only processes character nodes into glyph nodes and it's up to the user to provide more by manipulating the node lists. Of course there is still basic ligature building and kerning available but one can bypass that with other code.

In MkIV, when we deal with Type1 fonts, we try to get away from traditional tfm files and use afm files instead (indeed, we parse them using `lpeg`). The fonts are mapped onto Unicode. Awaiting extensions of math we only use tfm files for math fonts. Of course OpenType fonts are dealt with and this is where we find most Lua code in MkIV: implementing features. Much of that is a grey area but as part of the Oriental TEX project we're forced to deal with complex feature support, so that provides a good test bed as well as

some pressure for getting it done. Of course there is always the question to what extent we should follow the (maybe faulty) other programs that deal with font features. We're lucky that the Latin Modern and TEX Gyre projects provide real fonts as well as room for discussion and exploring these grey areas.

In parallel to writing this, I made a tracing feature for Oriental TEXer Idris so that he could trace what happened with the Arabic fonts that he is making. This was relatively easy because already in an early stage of MkIV some debugging mechanisms were built. One of its nice features is that on an error, or when one traces something, the results will be shown in a web browser. Unfortunately I have not enough time to explore such aspects in more detail, but at least it demonstrates that we can change some aspects of the traditional interaction with TEX in more radical ways.

Many users may be aware of the existence of so-called virtual fonts, if only because it can be a cause of problems (related to map files and such). Virtual fonts have a lot of potential but because they were related to TEX's own font data format they never got very popular. In LuaTEX we can make virtual fonts at runtime. In MkIV for instance we have a feature (we provide features beyond what OpenType does) that completes a font by composing missing glyphs on the fly. More of this trickery can be expected as soon as we have time and reason to implement it.

In pdfTEX we have a couple of font related goodies, like character expansion (inspired by Hermann Zapf) and character protruding. There are a few more but these had limitations and were suboptimal and therefore have been removed from LuaTEX. After all, they can be implemented more robustly in Lua. The two mentioned extensions have been (of course) kept and have been partially reimplemented so that they are now uniquely bound to fonts (instead of being common to fonts that traditional TEX shares in memory). The character related tables can be filled with Lua and this is what MkIV now does. As a result much TEX code could go away. We still use shape related vectors to set up the values, but we also use information stored in our main character database.

A likely area of change is math and not only as a result of the TEX gyre math project which will result in a bunch of Unicode compliant math fonts. Currently in MkIV the initialization already partly takes place using the character database, and so again we will end up with less TEX code. A side effect of removing encoding constraints (i.e. moving to Unicode) is that things get faster. Later this year math will be opened up.

One of the biggest impacts of opening up is the arrival of attributes. In traditional TEX only glyph nodes have an attribute, namely the font id. Now all nodes can have attributes, many of them. We use them to implement a variety of features that already were present in MkII, but used marks instead: color (of course including color spaces and transparency), inter-character spacing, character case manipulation, language dependent pre and post character spacing (for instance after colons in French), special font rendering such as outlines, and much more. An experimental application is a more advanced glue/penalty model with look-back and look-ahead as well as relative weights. This is inspired by the one good thing that xml formatting objects provide: a spacing and pagebreak model.

It does not take much imagination to see that features demanding processing of node lists come with a price: many of the callbacks that LuaTEX provides are indeed used and as a result quite some time is spent in Lua. You can add to that the time needed for handling font features, which also boils down to processing node lists. The second half of 2007 Taco and I spent much time on benchmarking and by now the interface between TEX and Lua (passing information and manipulating nodes) has been optimized quite well. Of course there's always a price for flexibility and LuaTEX will never be as fast as pdfTEX, but then, pdfTEX does not deal with OpenType and such.

We can safely conclude that the impact of LuaTEX on ConTEXt is huge and that fundamental changes take place in all key components: files, fonts, languages, graphics, MetaPost xml, verbatim and color to start with, but more will follow. Of course there are also less prominent areas where we use Lua based approaches: handling url's, conversions, alternative math input to mention a few. Sometime in 2009 we expect to start working on more fundamental typesetting related issues.

Roadmap

On the LuaTEX website `www.luatex.org` you can find a roadmap. This roadmap is just an indication of what happened and will happen and it will be updated when we feel the need. Here is a summary.

- merging engines
  Merge some of the Aleph codebase into pdfTEX (which already has ε-TEX) so that LuaTEX in dvi mode behaves like Aleph, and in pdf mode like pdfTEX. There will be Lua callbacks for file searching. This stage is mostly finished.
- OpenType fonts
  Provide pdf output for Aleph bidirectional functionality and add support for OpenType fonts. Allow Lua scripts to control all aspects of font loading, font definition and manipulation. Most of this is finished.

- tokenizing and node lists
  Use Lua callbacks for various internals, complete access to tokenizer and provide access to node lists at moments that make sense. This stage is completed.
- paragraph building
  Provide control over various aspects of paragraph building (hyphenation, kerning, ligature building), dynamic loading loading of hyphenation patterns. Apart from some small details these objectives are met.
- MetaPost (MPlib)
  Incorporate a MetaPost library and investigate options for runtime font generation and manipulation. This activity is on schedule and integration will take place before summer 2008.
- image handling
  Image identification and loading in Lua including scaling and object management. This is nicely on schedule, the first version of the image library showed up in the 0.22 beta and some more features are planned.
- special features
  Cleaning up of hz optimization and protruding and getting rid of remaining global font properties. This includes some cleanup of the backend. Most of this stage is finished.
- page building
  Control over page building and access to internals that matter. Access to inserts. This is on the agenda for late 2008.
- TEX primitives
  Access to and control over most TEX primitives (and related mechanisms) as well as all registers. Especially box handling has to be reinvented. This is an ongoing effort.
- pdf backend
  Open up most backend related features, like annotations and object management. The first code will show up at the end of 2008.
- math
  Open up the math engine parallel to the development of the TEX Gyre math fonts. Work on this will start during 2008 and we hope that it will be finished by early 2009.
- cweb
  Convert the TEX Pascal source into cweb and start using Lua as glue language for components. This will be tested on MPlib first. This is on the long term agenda, so maybe around 2010 you will see the first signs.

In addition to the mentioned functionality we have a couple of ideas that we will implement along the road. The first formal beta was released at tug 2007 in San Diego (usa). The first formal release will be at tug 2008 in Cork (Ireland). The production version will be released at EuroTEX in the Netherlands (2009).

Eventually LuaTEX will be the successor to pdfTEX (informally we talk of pdfTEX version 2). It can already be used as a drop-in for Aleph (the stable variant of Omega). It provides a scripting engine without the need to install a specific scripting environment. These factors are among the reasons why distributors have added the binaries to the collections. Norbert Preining maintains the linux packages, Akira Kakuto provides Windows binaries as part of his distribution, Arthur Reutenauer takes care of MacOSX and Christian Schenk recently added LuaTEX to MikTEX. The LuaTEX and MPlib projects are hosted at Supelec by Fabrice Popineau (one of our technical consultants). And with Karl Berry being one of our motivating supporters, you can be sure that the binaries will end up someplace in TEXLive this year.

Hans Hagen