LuaTEX 0.60: An overview of changes

Taco Hoekwater and Hartmut Henkel

Abstract

TEX Live 2010 will contain LuaTEX 0.60. This article gives an overview of the changes between this version and the version in last year's TEX Live.

Highlights of this release: CWEB code base, dynamic loading of lua modules, various font subsystem improvements including support for Apple `.dfont` font collection files, braced input file names, extended PDF Lua table, and access to the line breaking algorithm from Lua code.

1 General changes

Some of the changes can be organised into sections, but not all. So first, here are the changes that are more or less standalone.

- Many of the source files have been converted into CWEB. Early versions of LuaTEX were based on Pascal WEB, but by 0.40 all code had been hand-converted to C. The literate programming comments were kept, and the relevant sources have now been converted back into CWEB, reinstating the literate documentation.

  This change does not make LuaTEX a literate program in the traditional sense because the typical C source code layout with pairs of header & implementation files has been kept and no code reshuffling takes place. But it does mean that it is much easier to keep the source documentation up-to-date, and it is possible to create nicely typeset program listings with indices.
- There are now source repository revision numbers in the banner again, which is a useful thing to have while tracking down bugs. For example, the LuaTEX binary being used to write this article starts up with (except all on one line):

  ```
  This is LuaTeX,
    Version beta-0.60.1-2010042817 (rev 3659)
  ```
- The horizontal nodes that are added during line breaking now inherit the attributes from the nodes inside the created line. Previously, these nodes (`\leftskip` and `\rightskip` in particular) inherited the attributes in effect at the end of the (partial) paragraph because that is where line breaking takes place.
- All Lua errors now report file and line numbers to aid in debugging, even if the error happens inside a callback.
- LuaTEX can now use the embedded Kpathsea library to find Lua `require()` files, and will do so by default if the Kpathsea library is enabled by the format (as is the case in plain LuaTEX and the various LuaLATEX formats).
- The print precision for small numbers in Lua code (the return value of `tostring()`) has been improved.
- Of course there were lots of code cleanups and improvements to the reference manual.

2 Embedded libraries and other third-party inclusions

The following are changes to third-party code that for the most part should not need much explanation.

- MetaPost is now at version 1.211.
- Libpng is now at version 1.2.40.
- New SyncTEX code is imported from TEX Live.
- The Lua source file from the `luamd5` library (which provides the `md5.hexsuma` function) is now embedded in the executable. Previously, this file was missing completely.
- The Lua co-routine patch (`coco`) is now disabled on `powerpc-linux` because of crashes on that platform due to a bad upstream implementation.

2.1 Dynamic loading of lua modules

LuaTEX now has support for dynamic loading of external compiled Lua libraries.

As with other `require()` files, LuaTEX can and will use Kpathsea if the format allows it to do so. For this purpose, Kpathsea has been extended with a new file type: `clua`. The associated `texmf.cnf` variable is defined like this by default:

```
CLUAINPUTS = \
  .:$SELFAUTOLOC/lib/{$progname,$engine,}/lua//
```

which means that if your LuaTEX binary lives in

```
/opt/tex/texmf-linux-64/bin/
```

then your compiled Lua modules should go into the local directory, or in a tree below

```
/opt/tex/texmf-linux-64/bin/lib/lua
```

Be warned that not all available Lua modules will work. LuaTEX is a command line program, and on some platforms that makes it nearly impossible to use GUI-based extensions.

3 Font related

Lots of small changes have taken place in the font processing.

- The backend message

  ```
  cannot open Type 1 font file for reading
  ```

  now reports the name of the Type1 font file it was looking for.

- It is no longer possible for fonts from included PDF files to be replaced by or merged with the document fonts of the enveloping PDF.
- Support for Type 3 `.pgc` files has been removed. This is just for the `.pgc` format invented by Hàn Thế Thành; bitmapped PK files still work.
- For TrueType font collections (`.ttc` files), the used subfont name and its index id are now printed to the terminal, and if the backend cannot find the font in the `.ttc`, the run is aborted.
- It is now possible to use Apple `.dfont` font collection files. Unfortunately, in Snow Leopard (a.k.a. Mac OS X 10.6) Apple switched to a `.ttc` format that is not quite compatible with the Microsoft version of `.ttc`. As a result, the system fonts from Snow Leopard cannot be used in LuaTEX 0.60.
- Faster loading of large fonts via the `fontloader` library, and faster inclusion for subsetting in the backend.
- Two new entries in the `MathConstants` table have been added. Suppose the Lua math font loading code produces a Lua table named `f`, then in that table, you can set

```
f.MathConstants.FractionDelimiterSize
f.MathConstants.
        FractionDelimiterDisplayStyleSize
```

  These new fields allow proper setting of the size parameters for LuaTEX's `...withdelims` math primitives, for which there is no ready replacement in the OpenType MATH table.
- Artificially slanted or extended fonts now work via the PDF text matrix so that this also works for non-Type 1 fonts. In other words: the Lua `f.slant` and `f.extend` font keys are now obeyed in all cases.
- Another new key is allowed: `f.psname`. When set, this value should be the original PostScript font name of the font. In the PDF generation backend, fonts inside `.dfont` and `.ttc` collections are fetched from the archive using this field, so in those cases the key is required.
- A related change to the font name discovery used by the backend for storage into the PDF file structure: now it tries `f.psname` first, as that is much less likely to contain spaces than `f.fontname` (which is the field that 0.40 used). If there is no `f.psname`, it falls back to the old behaviour.
- Finally, Lua-loaded fonts now support the key `f.nomath` to speed up loading the Lua table in the normal case of fonts that do not provide OpenType MATH data.

4 'TEX'-side extensions and changes

LuaTEX is not actually TEX even though it uses an input language that is very similar, hence the quotes in this section's title. Some of the following items are new LuaTEX extensions, others are adjustments to pre-existing pdfTEX or Aleph functionality.

- The primitives `\input` and `\openin` now accept braced file names, removing the need for double quote escapes in case of files with spaces in their name.
- The `\endlinechar` can now be set to any value between 0 and 127.
- The new primitives `\aligntab` and `\alignmark` are aliases for the characters with the category codes of `&` and `#` in alignments, respectively.
- `\latelua` is now allowed inside leaders. To be used with care, because the Lua code will be executed once for each generated leader item.
- The new primitive `\gleaders` provides 'globally aligned' leaders. These leaders are aligned on one side of the main output box instead of to the side of the immediately enclosing box.
- From now on LuaTEX handles only 4 direction specifiers:
  - `TLT` (latin),
  - `TRT` (arabic),
  - `RTT` (cjk), and
  - `LTL` (mongolian).

  Other direction specifiers generate an error.
- The `\pdfcompresslevel` is now effectively fixed as soon as any output to the PDF file has occurred.
- `\pdfobj` has gained an extra optional keyword: `uncompressed`. This forces the object to be written to the PDF in plain text, which is needed for certain objects containing metadata.
- Two new token lists are provided: `\pdfxformattr` and `\pdfxformresources`, as an alternative to `\pdfxform` keywords.
- The new syntax

  `\pdfrefxform` [`width` ⟨*dimen*⟩] [`height` ⟨*dimen*⟩] [`depth` ⟨*dimen*⟩] ⟨*formref*⟩

  scales a single form object using similar principle as with `\pdfximage`: depth alone doesn't scale, it shifts vertically.
- Similarly,

  `\pdfrefximage` [`width` ⟨*dimen*⟩] [`height` ⟨*dimen*⟩] [`depth` ⟨*dimen*⟩] ⟨*imageref*⟩

  overrules settings from `\pdfximage` for this image only.

- The following obsolete pdfTEX primitives have been removed:
  - \pdfoptionalwaysusepdfpagebox
  - \pdfoptionpdfinclusionerrorlevel
  - \pdfforcepagebox
  - \pdfmovechars

  These were already deprecated in pdfTEX itself.

5 Lua table extensions

In most of the Lua tables that LuaTEX provides, only small changes have taken place, so they do not deserve their own subsections.

- A new callback, process_output_buffer, allows post-processing of \write text to a file.
- The callbacks hpack_filter, vpack_filter and pre_output_filter pass on an extra string argument for the current direction.
- fontloader.open() previously cleared some of the font name strings during load that it should not do.
- The new function font.id("tenrm") returns the internal id number for that font. It takes a bare control sequence name as argument.
- The os.name variable now knows about cygwin and kfreebsd.
- lfs.readlink("file") returns the content of a symbolic link (Unix only). This extension is intended for use in texlua scripts.
- lfs.shortname("file") returns the short (FAT) name of a file (Windows only). This extension is intended for use in texlua scripts.
- kpse.version() returns the Kpathsea version string.
- kpse.lookup(...) offers a search interface similar to the kpsewhich program, an example call looks like this:

```
kpse.set_program_name('luatex')
print(kpse.lookup('plain.tex',
                  { ["format"] = "tex",
                    ["all"] = true,
                    ["must-exist"] = true }))
```

5.1 The node table

In the verbatim code below, n stands for a userdata node object.

- node.vpack(n) packs a list into a vlist node, like \vbox.
- node.protrusion_skippable(n) returns true if this node can be skipped for the purpose of protrusion discovery. This is useful if you want to (re)calculate protrusion in pure Lua.
- node.dimensions(n) returns the natural width, height and depth of a (horizontal) node list.
- node.tail(n) returns the tail node of a node list.
- Each glyph node now has three new virtual read-only fields: width, height, and depth. The values are the number of scaled points.
- glue_spec nodes now have an extra boolean read-only field: writable.
  Some glue specifications can be altered directly, but certain key glue specifications are shared among many nodes. Altering the values of those is prohibited because it would have unpredictable side-effects. For those cases, a copy must be made and assigned to the parent node.
- hlist nodes now have a subtype to distinguish between hlists generated by the paragraph breaking, explicit \hbox commands, and other sources.
- node.copy_list(n) now allows a second argument. This argument can be used to copy only part of a node list.
- node.hpack(n) now accepts cal_expand_ratio and subst_ex_font modifiers. This feature helps the implementation of font expansion in a pure Lua paragraph breaking code.
- node.hpack(n) and node.vpack(n) now also return the 'badness' of the created box, and accept an optional direction argument.

5.2 The pdf table

- The new functions pdf.mapfile("...") and pdf.mapline("...") are aliases for the corresponding pdfTEX primitives.
- pdf.registerannot() reserves a PDF object number and returns it.
- The functions pdf.obj(), pdf.immediateobj(), and pdf.reserveobj() are similar to the corresponding pdfTEX primitives. Full syntax details in the LuaTEX reference manual.
- New read-write string keys:
  - pdf.catalog in the Catalog dictionary.
  - pdf.info in the Info dictionary.
  - pdf.names in the Names dictionary referenced by the Catalog object.
  - pdf.trailer in the Trailer dictionary.
  - pdf.pageattributes in the Page dictionary.
  - pdf.pageresources in the Resources dictionary referenced by the Page object.
  - pdf.pagesattributes in the Pages dictionary.

5.3 The tex table

Finally, there are some extensions to the `tex` table that are worth mentioning.

- `tex.badness(f,s)` interfaces to the 'badness' internal function. (By accident, this disables access to the `\badness` internal parameter. This will be corrected in a future LuaTEX version.)
- `tex.sp("1in")` converts Lua-style string units to scaled points.
- `tex.tprint(...,...)` is like a sequence of `tex.sprint(...)` calls.
- `tex.shipout(n)` ships out a constructed box.
- `tex.nest[]` and `tex.nest.ptr` together allow read-write access to the semantic nest (mode nesting). For example, this prints the equivalent of `\prevdepth` at the current mode nesting level:

  ```
  print (tex.nest[tex.nest.ptr].prevdepth)
  ```

  `tex.nest.ptr` is the current level, and lower numbers are enclosing modes.

  Each of the items in the `tex.nest` array represents a mode nesting level and has a set of virtual keys that be accessed both for reading and writing, but you cannot change the actual `tex.nest` array itself. The possible keys are listed in the LuaTEX reference manual.
- `tex.linebreak(n, ...)` supports running the paragraph breaker from pure Lua. The second argument specifies a (potentially large) table of line breaking parameters: the parameters that are not passed explicitly are taken from the current typesetter state.

  The exact keys in the table are documented in the reference manual, but here is a simple yet complete example of how to run line breaking on the content of `\box0`:

  ```
  \setbox0=\hbox to \hsize{\input knuth }
  \startluacode
  local n = node.copy_list(tex.box[0].list)
  local t = node.tail(n)
  local final = node.new(node.id('glue'))
  final.spec = node.new(node.id('glue_spec'))
  final.spec.stretch_order = 2
  final.spec.stretch = 1
  node.insert_after(n,t, final)
  local m = tex.linebreak(n,
            { hangafter = 2,
              hangindent = tex.sp("2em")})
  local q = node.vpack(m)
  node.write(q)
  \stopluacode
  ```

  The result is:

> Thus, I came to the conclusion that the designer of a new system must not only be the implementer and first large–scale user; the designer should also write the first user manual. The separation of any of these four components would have hurt TEX significantly. If I had not participated fully in all these activities, literally hundreds of improvements would never have been made, because I would never have thought of them or perceived why they were important. But a system cannot be successful if it is too strongly influenced by a single person. Once the initial design is complete and fairly robust, the real test begins as people with many different viewpoints undertake their own experiments.

6 Summary

All in all, there are not too many incompatible changes compared to LuaTEX 0.40, and the LuaTEX project is progressing nicely.

LuaTEX beta 0.70 will be released in the autumn of 2010. Our current plans for that release are: access to the actual PDF structures of included PDF images; a partial redesign of the mixed direction model; even more access to the LuaTEX internals from Lua; and probably some more ...

◇ Taco Hoekwater and Hartmut Henkel
http://luatex.org