OpenType fonts in LuaTEX

Taco Hoekwater
http://luatex.org

Abstract

Since the start of February 2007, LuaTEX has supported the use of OpenType fonts directly, without the need for separate metrics and font map files. This talk will explain and demonstrate how this works in practice.

1 Introduction

This is an updated version of the paper that was in the preprint. There has been considerable progress in the time between the preprint (early April) and now (early August). The current paper documents the state of affairs in LuaTEX 0.10, the first public beta.

2 OpenType fonts in LuaTEX

If you want to do typesetting with TEX, you have to get the required font metric information from somewhere. METAFONT- or fontinst-based fonts typically come as a set of TFM and VF files, and for those, LuaTEX behaves in a way that is backward compatible with any other traditional version of TEX.

But loading metrics for OpenType (`.otf`) and TrueType (`.ttf` and `.ttc`) fonts can also be done through a Lua extension interface, in which case there is no need for TFM, VF, ENC, and MAP files. Instead, you (or more precisely, a macro package writer) has to write a bit of Lua code.

There are two separate parts to this process, that will be explained in turn in the next paragraphs:

1. You have to make LuaTEX use the Lua extension interface instead of the compatibility mode metrics loading, by setting up a Lua callback.
2. You have to write the necessary Lua code to make it possible for LuaTEX to use the OpenType fonts you have installed.

3 Font definitions through Lua callbacks

Installing 'callbacks' is one of the most important concepts in LuaTEX. A callback is what we call the situation whereby LuaTEX is instructed to run a user-supplied Lua function instead of a bit of internal compiled code. A few dozen of these interception points are defined at this time, and they have all been given names.

You install a callback by connecting a Lua function to one of these names. For this purpose, there is a predefined 'register' Lua function provided. The most relevant callback for font definitions is named 'define_font', and it could be set up like so:

```
\directlua0 {
 function read_font (name, size, fontid)
   local file = kpse.find_file (name, 'tfm')
   local metrics = font.read_tfm (file, size)
   return metrics
 end

 callback.register('define_font', read_font)
}
```

This example first defines a function to do the work (`read_font`), and then registers that function as a callback. The function does essentially the same as what TEX would have done without any callback. It uses the functions `kpse.find_file` and `font.read_tfm`, which are predefined helper functions.

When LuaTEX next runs into a `\font` command, it will gather the user-supplied font name and size specification, and pass those values on to the Lua function `read_font` as the first two arguments. The task of `read_font` is to create a data structure (in Lua this is called a 'table') that contains the metric information needed for typesetting in the font `name` loaded at size `size`.

The internal structure of the Lua table that is to be returned by `read_font` is explained in detail in the LuaTEX manual. Fortunately for the length of the example, that structure is a super-set of the structure returned by `font.read_tfm`, so it can just be passed along without further manipulation.

In the example you can see that there is a third argument to `read_font`, ignored in this case. LuaTEX also passes the internal id number of the font that is going to be defined. This is because, in macro packages, it is not abnormal for the same font to be defined more than once using the same `name` and `size` specification, so instead of returning a Lua table defining the metrics for a font, it is also legal to return just a number, referencing the `fontid` of an already defined font. That way, you could set up a lookup table of already defined fonts.

4 Handling OpenType fonts

There is a Lua module included in LuaTEX that can be used to read a font's metrics from the disk, using the font reading code from the open source program FontForge.

The contents of this module are available in the Lua table named `fontforge`. Using it, the basic way to get the metric information is like this:

```
function load_font (filename)
  local metrics = nil
  local font = fontforge.open(filename)
  if font then
     metrics = fontforge.to_table(font)
  end
  return metrics
end
```

This code first loads the font into program memory with `fontforge.open`, and then converts it to a Lua table by calling `fontforge.to_table`.

The font file is parsed and partially interpreted by the font loading routines from FontForge. The file format can actually be any one of OpenType, TrueType, TrueType Collection, CFF, or Type 1.

There are a few important advantages to using this approach with a dedicated Lua module, compared to having a single dedicated helper function to read an OpenType font file:

- The internal font encoding is automatically processed, so that the returned `metrics` table also contains the Unicode encoding information for all the included glyphs.
- Many OpenType features are pre-processed into a format that is easier to handle than just the bare feature tables would be.
- And looking at it from the other side: it is still possible to completely alter any feature you want to change, as nothing at all is hardwired in the executable.
- PostScript-based fonts do not store the character height and depth in the font file (in Type 1 fonts, this information is in the AFM file, in CFF fonts it is not present at all). For CFF fonts, this means that the character bounding box has to be properly calculated, a task that is handled internally by FontForge.
- In the future, it may be interesting to allow Lua scripts access to the actual font program, perhaps even creating or changing a font file itself.

However, there is also a downside: the data structure of the table returned by the OpenType reading routines is very low-level and very close to the internal format used by FontForge itself.

This means that it is not compatible with the table structure required by the font definition callback, so some modifications to the structure are needed before the table can be passed on to LuaTEX proper. This area is still under development. We plan to provide a set of helper functions for this task eventually but for the moment, this has to be done by Lua code you write yourself.

To finish off this introduction, here is a small peek into the table returned by `fontforge.open`. What follows is a human-readable representation of the ligature glyph for 'fi' in the font lmroman10-regular.otf:

```
{
 ["boundingbox"]={ 27, 0, 527, 705 },
 ["lookups"]={
  ["ls_l_10_s"]={
   {
    ["specification"]={
     ["char"]="f_i",
     ["components"]="f i",
    },
    ["type"]="ligature",
   },
  },
 },
 ["name"]="f_i",
 ["unicodeenc"]=64257,
 ["width"]=556,
}
```