LuaTEX: PDF merging

Hans Hagen

1 Introduction

It is tempting to add more and more features to the backend code of the engine but it is not really needed. Of course there are features that can best be supported natively, like including images. In order to include PDF images in LuaTEX the backend uses a library (xpdf or poppler) that can load a page from a file and embed that page into the final PDF, including all relevant (indirect) objects needed for rendering. In LuaTEX an experimental interface to this library is included, tagged as `epdf`. In this chapter I will spend a few words on my first attempt to use this new library.

2 The library

The interface is rather low level. I got the following example from Hartmut Henkel, who is responsible for the LuaTEX backend code and this library.

```
local doc = epdf.open("luatexref-t.pdf")
local cat = doc:getCatalog()
local pag = cat:getPage(3)
local box = pag:getMediaBox()

local w = pag:getMediaWidth()
local h = pag:getMediaHeight()
local n = cat:getNumPages()
local m = cat:readMetadata()

print("nofpages: ", n)
print("metadata: ", m)
print("pagesize: ", w .. " * " .. h)
print("mediabox: ",box.x1,box.x2,box.y1,box.y2)
```

As you see, there are accessors for each interesting property of the file. Of course such an interface needs to be extended when the PDF standard evolves. However, once we have access to the so-called catalog, we can use regular accessors to the dictionaries, arrays and other data structures. So, in fact we don't need a full interface and can draw the line somewhere.

There are a couple of things that you normally do not want to deal with. A PDF file is in fact a collection of objects that form a tree and each object can be reached by an index using a table that links the index to a position in the file. You don't want to be bothered with that kind of housekeeping. Some data in the file, like page objects and annotations, are organized in a tree form that one does not want to access in that form, so again we have something that benefits from an interface. But the majority of the objects are simple dictionaries and arrays. Streams (these hold the document content, image data, etc.) are normally not of much interest, but the library provides an interface as you can bet on needing it someday. The library also provides ways to extend the loaded PDF file. I will not discuss that here.

Because in ConTEXt we already have the `lpdf` library for creating PDF structures, it makes sense to define a similar interface for accessing PDF. For that I wrote a wrapper that will be extended in due time (read: depending on needs). The previous code now looks as follows:

```
local doc = epdf.open("luatexref-t.pdf")
local cat = doc.Catalog
local pag = cat.Pages[3]
local box = pag.MediaBox

local llx, lly, urx, ury
  = box[1], box[2] box[3], box[4]

local w = urx - llx -- or: box.width
local h = ury - lly -- or: box.height
local n = cat.Pages.size
local m = cat.Metadata.stream

print("nofpages: ", n)
print("metadata: ", m)
print("pagesize: ", w .. " * " .. h)
print("mediabox: ", llx, lly, urx, ury)
```

If we write code this way we are less dependent on the exact API, especially because the `epdf` library uses methods to access the data and we cannot easily overload method names in there. When you look at the `box`, you will see that the natural way to access entries is using a number. As a bonus we also provide the `width` and `height` entries.

3 Merging links

It has always been on my agenda to add the possibility to carry the (link) annotations with an included page from a document. This is not that much needed in regular documents, but it can be handy when you use ConTEXt to assemble documents. In any case, such a merge has to happen in a way that does not interfere with other links in the parent document. Supporting this in the engine is not an option as each macro package follows its own approach to referencing and interactivity. Also, demands might differ and one would end up with a lot of (error prone) configurability. Of course we want scaled pages to behave well too.

Implementing the merge took about a day and most of that time was spent on experimenting with the `epdf` library and making the first version of the wrapper. I definitely had expected to waste more time on it. So, this is yet another example of an

extension that is quite doable in the Lua–TEX mix. Of course it helps that the ConTEXt graphic inclusion code provides enough information to integrate such a feature. The merge is controlled by the interaction key, as shown here:

```
\externalfigure[somefile.pdf][page=1,scale=700,
                              interaction=yes]
\externalfigure[somefile.pdf][page=2,scale=600,
                              interaction=yes]
```

You can fine-tune the merge by providing a list of options to the interaction key but that's still somewhat experimental. As a start the following links are supported.

- internal references by name (often structure related)
- internal references by page (like on tables of contents)
- external references by file (optionally by name and page)
- references to URIs (normally used for web pages)

When users like this functionality (or when I really need it myself) more types of annotations can be added although support for JavaScript and widgets doesn't make much sense. On the other hand, support for destinations is currently somewhat simplified but at some point we will support the relevant zoom options.

The implementation is not that complex:

- check if the included page has annotations
- loop over the list of annotations and determine if an annotation is supported (currently links)
- analyze the annotation and overlay a button using the destination that belongs to the annotation

Now, the reason why we can keep the implementation so simple is that we just map onto existing ConTEXt functionality. And, as we have a rather integrated support for interactive actions, only a few basic commands are involved. Although we could do that all in Lua, we delegate this to TEX. We create a layer that we put on top of the image. Links are put onto this layer using the equivalent of:

```
\setlayer
 [epdflinks]
 [x=...,y=...,preset=leftbottom]
 {\button
  [width=...,height=...,offset=overlay,frame=off]
  {}% no content
  [...]}}
```

The `\button` command is one of those interaction-related commands that accepts any action-related directive. In this first implementation we see the following destinations show up:

```
somelocation
url(http://www.pragma-ade.com)
file(somefile)
somefile::somelocation
somefile::page(10)
```

References to pages become named destinations and are later resolved to page destinations again, depending on the configuration of the main document. The links within an included file get their own namespace so (hopefully) they will not clash with other links.

We could use lower-level code which is faster but we're not talking of time-critical code here. At some point I might optimize the code a bit but for the moment this variant gives us some tracing options for free. Now, the nice thing about using this approach is that the already existing cross-referencing mechanisms deal with the details. Each included page gets a unique reference so references to not-included pages are ignored simply because they cannot be resolved. We can even consider overloading certain types of links or ignoring named destinations that match a specific pattern. Nothing is hard coded in the engine so we have complete freedom in doing that.

4 Merging layers

When including graphics from other applications it might be that they have their content organized in layers (that can then be turned on or off). So it will be no surprise that merging layer information is on the agenda: first a straightforward inclusion of optional content dictionaries, but it might make sense to parse the content stream and replace references to layers by those that are relevant in the main document. Especially when graphics come from different sources and layer names are inconsistent some manipulation might be needed, so maybe we need more detailed control. Implementing this is no big deal and mostly a matter of figuring out a clean and simple user interface.

◇ Hans Hagen
Pragma ADE
The Netherlands
http://luatex.org