PDF/A-1a in ConTEXt MkIV

Abstract
I present some considerations on electronic document archiving and how ConTEXt MkIV supports the ISO Standard 19500-1 Level A Conformance (PDF/A-1a:2005), an ISO standard for long-term document archiving.

Keywords
LuaTeX, ConTeXt MkIV, PDF/A, color, font.

Introduction

In this paper I will briefly talk about the ISO Standard PDF/A-1 and how ConTEXt MkIV tries to adhere to its requirements by showing some practical examples. About the typographic style of this paper: I will follow these simple rules: I will avoid footnotes and citations on running text, and I will try to limit lists (e.g. only itemize and enumerate) and figures; the last section before the References one will collect all citations.

The PDF/A-1 ISO Standard

Probably one of the best known PDF versions is PDF 1.4 (around 2001, almost ten years ago) maybe because the companion Acrobat 5.0 was a robust program and the PDF Reader was freely available for several platforms both as a program and as plug in for browsers. We keep having a huge amount of electronic documents that are in PDF 1.4, hence we should not be surprised if Adobe pushed it as reference for document archiving. What follows is a verbatim copy from http://www.digitalpreservation.gov/formats/fdd/fdd000125.shtml and it's a good description:

> PDF/A-1 is a constrained form of Adobe PDF version 1.4 intended to be suitable for long-term preservation of page-oriented documents for which PDF is already being used in practice. The ISO standard [ISO 19005-1:2005] was developed by a working group with representatives from government, industry, and academia and active support from Adobe Systems Incorporated. Part 2 of ISO 19005 (as of September 2010, an ISO Draft International Standard) extends the capabilities of Part 1. It is based on PDF version 1.7 (as defined in ISO 32000-1) rather than PDF version 1.4 (which is used as the basis of ISO 19005-1).
>
> PDF/A attempts to maximize device independence, self-containment, self-documentation. The constraints include: audio and video content are forbidden, JavaScript and executable file launches are prohibited, All fonts must be embedded and also must be legally embeddable for unlimited, universal rendering, colorspaces specified in a device-independent manner, encryption is disallowed, use of standards-based metadata is mandated.
>
> The PDF/A-1 standard defines two levels of conformance: conformance level A satisfies all requirements in the specification; level B is a lower level of conformance, 'encompassing the requirements of this part of ISO 19005 regarding the visual appearance of electronic documents, but not their structural or semantic properties'.

In essence the standard wants to ensure that every typographic element, from the low level character to the high level logical structure is unambiguously defined and unchangeable — and it does, it achieves its purpose: every character must be identified by a Unicode id, which is an international standard, every color must be device independent by means of a color profile or output intent, there must be precise meta data informations for classifications and the document must have a logical structure described by a (possible ad-hoc) markup language.

Unfortunately the PDF version 1.4 is quite old: animations and 3D pictures cannot be embedded, the font format cannot be OpenType, JavaScript programs are not permitted at all, even if they don't modify the document in any way as, for example, a calculator. Ten years ago it was very important to guarantee that the document would always be printed as intended, nowadays screen is slowly replacing paper and animations play a fundamental role: PDF/A-1 is good for paper but less than optimal for 'electronic paper'.

PDF/A-1a in ConTEXt MkIV

Given that it is still under heavy development, ConTEXt

MkIV has the opportunity to be developed on two fronts: the 'low level' luaTEX (CWEB code and Lua primitives) and the 'high level' macros that build the format itself. One of this year's results is the implementation of 'tagged PDF', the Adobe document markup language for PDF documents, and the development of color macros for the PDF/X specifications. As a consequence, it was possible to use these results to test some real code for producing PDF/A-1a compliant documents. Let's start with an example explained step-by-step.

```
%% Debug
\enabletrackers[backend.format,
                backend.variables]
%% For PDF/A
\setupbackend[
format={pdf/a-1a:2005},
profile={default_cmyk.icc,
         default_rgb.icc,default_gray.icc},
intent={%
ISO coated v2 300\letterpercent\space (ECI)}
]
%% Tagged PDF
%% method=auto ==> default tags by Adobe
\setupstructure[state=start,method=auto]

\definecolor[Cyan][c=1.0,m=0.0,y=0.0,k=0.0]
\starttext
\startchapter[title={Test}]
\startparagraph
\input tufte
%% Some ConTeXt env. are already mapped:
%% colors
\color[red]{OK}
\color[Cyan]{OK}
%% figures
\externalfigure[rgb-icc-srgb.jpg]
               [width=0.4\textwidth]
\stopparagraph
%% Natural tables
\bTABLE
\bTR\bTD 1 \eTD \bTD 2 \eTD \eTR
\bTR \bTD[nx=2] 3 \eTD\eTR
\eTABLE
\stopchapter
\stoptext
```

As usual the file is processed with

```
#>context test.tex
```

and it doesn't hurt to enable some debug information with

```
\enabletrackers[backend.format,
                backend.variables]
```

Enable the PDF/A-1a

To enable PDF/A-1a we must setup the backend with the appropriate variant of PDF/A. From the very beginning ConTEXt has had a backend system that permits to use almost the same macro-format for different outputs (i.e. DVI and PDF), and with luaTEX this system is increasingly enhanced, as we'll see later on.
With

```
format={pdf/a-1a:2005}
```

we select the 1a variant of PDF/A standard and the label is mandatory because it also puts some default meta data into the output (see `lpdf-pda.xml`; a complete list of formats is currently in `lpdf-fmt.lua` and also as a Lua table `lpdf.formats`).

Next comes the colors part, and we must pay attention here. The key concept is:

every color must be independent of any device.

Usually in a PDF we have two sources for colors: the colors specified by the author, e.g. something like `\definecolor[orange][r=1.0,g=0.5,b=0.0]`, and the images. The most used color spaces `DeviceGray`, `DeviceRGB`, `DeviceCMYK` are device dependent because the reproduction of a color from these color spaces *depends* on the particular output device, and the real output devices are all different due both to the different nature (screen vs. printer, for example) and different technologies (CRT vs. LCD screen, or inkjet vs laser printer, for example). Every device can be classified by means of a *color profile* which maps an input color (rgb, cmyk or gray) to an *independent color space*: such maps ensure that each device will correctly reproduce the color, and also the independent color space permits to compare colors from different color spaces.

With

```
profile={default_cmyk.icc,
         default_rgb.icc,default_gray.icc},
```

we associate all the document colors with the corresponding color profile by mean of a filename (the file `colorprofiles.xml` has a list of predefined profiles). Be careful here: it's wrong to associate a rgb color space with a cmyk profile, and not all profiles are good, especially those for printing. Moreover PDF/A-1a allows only profiles having version 3 or below.

There is a second way to specify colors, and it's a bit complicated. We must specify that all the colors *without profile* are intended to be used with a common output profile, i.e. we must impose an *output intent*: this is the meaning of

```
intent={%
ISO coated v2 300\letterpercent\space (ECI)}
```

which is a cmyk profile for coated paper. Note that we are using a name and not a filename to avoid clashing with the values of the `profile` key.
By doing so we accept these implicit limitations and color space conversions:

- if the output intent is a cmyk profile then the document can have only cmyk and gray colors;
- if the output intent is a rgb profile then the document can have only rgb and gray colors;
- if the output intent is a gray profile then the document can have only gray colors.

They are reasonable: in general we cannot use a rgb color with a cmyk profile because there are rgb colors without equivalent cmyk ones (that is to say that screens display more colors than printers). We can convert a gray color to rgb or cmyk because usually gray color spaces are a subset of the former (otherwise we have a really poor device). It's not an error if we specify both profiles and output intent: at least if all color spaces have their own profiles, as in the example, then the output intent is simply ignored by a PDF/A compliant PDF reader.

Finally the images: we must be sure that every image has its color profile — and this can be a bit complicated.

In the following example, `rgb-noprofile.jpg` is a jpeg image with a RGB color space and without a color profile:

```
\setupbackend[
format={pdf/a-1a:2005},%level=0,
profile={default_cmyk.icc,
         default_rgb.icc,default_gray.icc},
]
\setupstructure[state=start]
\starttext
\startchapter[title={Test}]
\startparagraph
\externalfigure[rgb-noprofile.jpg]
             [width=0.4\textwidth]
\stopchapter
\stoptext
```

The luatex program loads the image, it wraps it in a `/XObject`, and sets its ColorSpace to DeviceRGB:

```
<<
/Type /XObject
/Subtype /Image
/Width 640
/Height 400
/BitsPerComponent 8
/Length 13238
/ColorSpace /DeviceRGB
/Filter /DCTDecode
>>
stream...endstream
```

This is a valid PDF/A-1a document, but if we delete the `default_rgb.icc` profile

```
profile={default_cmyk.icc,default_gray.icc},
```

then the resulting PDF is an *invalid* PDF/A. We should not be surprised: there is color space which is device dependent and hence we cannot guarantee the correct reproduction of the colors.

In the next example we use a rgb image *with a valid color profile*:

```
\setupbackend[
  format={pdf/a-1a:2005},%level=0,
  rofile={default_cmyk.icc,default_gray.icc}]
\setupstructure[state=start]
\starttext
\startchapter[title={Test}]
\startparagraph
\externalfigure[rgb-icc-srgb.jpg]
               [width=0.4\textwidth]
\stopchapter
\stoptext
```

For the same reason seen before, this PDF is still an invalid PDF/A: the image is again wrapped in a `/XObject` with a `/DeviceRGB` color space — but this time it's not correct: the image has its own profile and hence its colors are device independent. If we add a rgb profile we have again a valid PDF/A:

```
profile={default_cmyk.icc,
default_rgb.icc,default_gray.icc},
```

but this is dangerous because we don't know if it's correct for the image and also in this way *all* the rgb color spaces of others images are associated to this specific profile.

To remedy this situation, I present here a practical solution that relies on the MagickWand suite which is available for free for Windows, Linux and Mac platforms. The first step is to verify if the image has a profile:

```
#>gm identify -verbose rgb-icc-srgb.jpg
 :
 Profile-color: 3144 bytes
 :
```

The second step is to save the profile:

```
#>gm convert  rgb-icc-sRGB_v4_ICC.jpg  sRGB.icc
```

and the last step is to build a /XObject with the appropriate color space. This is a bit tricky, but fundamentally we mimic the behavior of luatex with ConTEXt MkIV. I will show only an example for a jpeg image with a /DeviceRGB color space:

```
%% rgb-icc-srgb.pdf
\pdfminorversion4
\starttext\startTEXpage%
\startluacode
local a=img.scan{filename="rgb-icc-srgb.jpg"}

tex.sprint(tex.ctxcatcodes,
 string.format(
  "\\startfoundexternalfigure{\%ssp}{\%ssp}",
  a.width,a.height))
 local icc_ref = pdf.immediateobj("streamfile",
  "srgb.icc",
  " /Alternate /DeviceRGB\n" ..
  "/Filter /FlateDecode\n/N 3")

local icc_dict_ref = pdf.immediateobj(
  string.format("[ /ICCBased \%d 0 R ]\n",
                icc_ref) )

a=img.new{filename="rgb-icc-srgb.jpg",
  colorspace=icc_dict_ref}

a=img.immediatewrite(a)

node.write(img.node(a))

tex.sprint(tex.ctxcatcodes,
  "\\stopfoundexternalfigure")
\stopluacode%
\stopTEXpage\stoptext
```

As we can see the XObject has now an ICCBased color space:

```
15 0 obj
<<
/Alternate /DeviceRGB
/Filter /FlateDecode
/N 3
/Length 3144
>>
stream...endstream
16 0 obj
[ /ICCBased 15 0 R ]
endobj
17 0 obj
<<
/Type /XObject
/Subtype /Image
/Width 640
/Height 400
/BitsPerComponent 8
/Length 9948
/ColorSpace 16 0 R
/Filter /DCTDecode
>>
stream...endstream
```

Once the image with the correct color space is wrapped in a PDF file (rgb-icc-srgb.pdf in this case), we can use it in our documents:

```
\setupbackend[
format=[{pdf/a-1a:2005},%level=0,
    profile={default_cmyk.icc,default_gray.icc},
      ]
\setupstructure[state=start]
\starttext
\startchapter[title={Test}]
\startparagraph
\externalfigure[rgb-icc-srgb.pdf]
               [width=0.4\textwidth]
\stopchapter
\stoptext
```

which is again a valid PDF/A.

Tagged PDF

Next we must enable the tagging system with \setupstructure[state=start,method=auto]. ConTEXt MkIV permits the author to define his own document markup language (the tags used inside the PDF document) but of course we also need the associated TEX macros. This naturally needs to start with a sort of XML document:

```
\setupstructure[state=start,method=none]
\starttext
\startelement[document]
\startelement[chapter]
opes
\startelement[p]\input ward\stopelement \par
\stopelement
\stopelement
\stoptext
```

The internal tag names are <document>, <chapter> and <p> as we see in fig. 1 from Acrobat 9.0, but we still need to put the appropriate typographic elements into the PDF.

In the context of PDF/A, a validation program expected the tags as defined by Adobe and this leads to some 'syntactic sugar' macros, i.e instead of

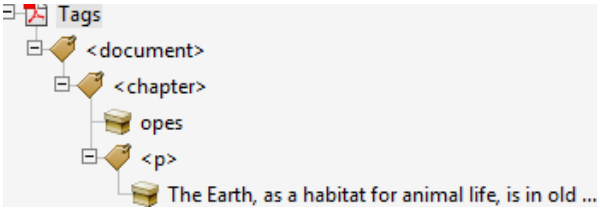

Figure 1 The tags structure of a simple document

`\startelement[chapter]...\stopelement`

it's better to use

`\startchapter[title={Test}]...\stopchapter`

which puts the correct tags and also typesets the chapter title Test as expected.

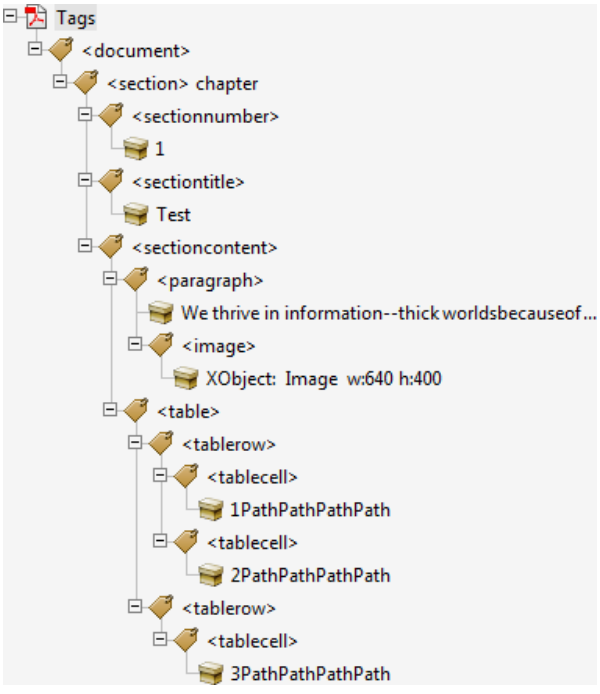

Figure 2 The tag structure of complex document

The complete list of tags can be found in `strc-tag.mkiv` and of course ConTEXt MkIV permits to redefine the default mapping. In fig. 2 our document shows that ConTEXt MkIV had already mapped some predefined typographic objects like figures and tables to the appropriate tags.

We can use this mechanism to embed an XML document into a tagged PDF document, which opens quite interesting perspectives, but we can also start from a 'structured TEX' document and end into an XML one, and this is more interesting because it's a matter of backend only — and because it's already implemented:

```
\setupbackend[export=yes]
\setupstructure[state=start,method=none]
\starttext
\startelement[document]
\startelement[chapter][title=Test]
opes
\startelement[p]\input ward\stopelement \par
\stopelement
\stopelement
\stoptext
```

produces a `<tex-file>.export` like this (original XML spaces are not preserved in this listing)

```
<?xml version='1.0' standalone='yes' ?>
<!-- input filename   : test-2            -->
<!-- processing date  : 10/09/10 15:28:48 -->
<!-- context version  : 2010.09.24 11:40  -->
<!-- exporter version : 0.10              -->
<document language='en'
   file='test-2' date='10/09/10 15:29:04'
   context='2010.09.24 11:40'
   version='0.10'>
<chapter title="Test">opes
<p>
The Earth, as a habitat for animal life, is
in old age and has a fatal illness. Several,
in fact. It would be happening whether humans
had ever evolved or not. But our presence is
like the effect of an old-age patient who
smokes many packs of cigarettes per day
------ and we humans are the cigarettes.
</p>
</chapter>
</document>
```

Fonts and encoding

In the previous subsection we have seen that with simple macros we can have a *valid* (i.e validated by Acrobat 9.0) PDF/A-1a PDF document. We still didn't talk about fonts.

The default fonts used by ConTEXt MkIV are the OpenType version of LatinModern, and, as of now, they cannot be embedded into PDF/A documents because OpenType isn't supported in version 1.4; this is not a problem because, in essence, ConTEXt MkIV strips the OpenType part and embeds a valid Type1 or TrueType font. Given an OpenType font, ConTEXt MkIV is also able to map each glyph to its Unicode `id`, so even this side is not problematic.

Unfortunately, it's already known that typesetting mathematics with the Computer Modern and Latin

Modern fonts easily leads to invalid PDF/A documents due to misleading dimensional information of some fonts. As widely noted by C. Beccari, just the simple `$a\not=b$` invalidates the whole document, due the wrong dimension of the `\not` sign (it has BoundingBox=(139,139,-960,775) hence a width equal to zero). What are the solutions? There are two of them, both unsatisfactory:

1. choose another (valid) math family;
2. make a high resolution (more than 300dpi) bitmap of each invalid formula.

Of course it's possible to edit the fonts, but it's not a general solution: there are limitations due to copyright and we should embed a modified copy of the font that differs from the original version — an error prone situation because modifications of PDF/A-1a document are permitted, and an editor can use the system fonts. The problem remains even if ConTEXt MkIV can patch the font on the fly. A way out is the complete embedding of the patched fonts, so that the editor uses the document fonts, but it's not a robust solution — some editors can still use the original system fonts.

Conclusion

The PDF/A-1a is a good standard for document archiving: it's a complete *Page Description Language*, it relies on Unicode which is also a good *Character Language* and on Type1 and TrueType as *digital typography formal language*; it has also a good *Document Markup Language*. The binary electronic format and the digital signature for detection and prevention of document modifications complete the picture. The restrictions (e.g. profiles for colors) together with a freely available PDF/A-1a PDF reader lead to a concrete self-containment format.

PDF/A-1a support in ConTEXt MkIV is still experimental: it needs more tests, but programming in luaTEX is simpler than in pdfTEX, and the 1.4 is a well known PDF version. The color management can probably be improved by permitting to specify a color and its profile for a specific object and not for the whole document, as it currently is.

On the other hand, the model of PDF/A-1 is the traditional paper. Omitting animations and 3D pictures is questionable and perhaps also scripting languages should be permitted if they don't modify the document.

The ISO standard is not freely available and the PDF/A-1a validators are complex to implement and usually expensive commercial products; this is an obstacle for the diffusion of PDF/A.

Notes on References

For the first section, some informations on PDF/A-1 are at Wikipedia [1], the techdoc at [2], and [4]. Very useful are also the references of C. Beccari's paper at [9]. An interesting use of JavaScript in PDF is [3].

For the second section, the ConTEXt wiki [6] has some terse informations, because the code is the ultimate reference. Tagged PDF is described in the version of `hybrid.pdf` [7] that is part of '*Proceedings of the 4th ConTEXt meeting*' [8](to be published). For ICC profiles a good starting point is [5]; the problems about fonts are described by C. Beccari in [9] and [10].

References

All links were verified between 2010.10.19 and 2010.10.22.

[1] http://en.wikipedia.org/wiki/PDF/A.
[2] http://www.pdfa.org/doku.php?id=pdfa:en:techdoc
[3] www.tug.org/applications/pdftex/calculat.pdf.
[4] http://www.digitalpreservation.gov/formats/fdd/fdd000125.shtml.
[5] http://en.wikipedia.org/wiki/ICC_profile.
[6] http://wiki.contextgarden.net/PDFX.
[7] http://www.pragma-ade.com/general/manuals/hybrid.pdf
[8] http://meeting.contextgarden.net/2010/talks/
[9] http://www.guit.sssup.it/downloads/Beccari_Pdf_archiviabile.pdf
[10] http://dw.tug.org/pracjourn/2010-1/beccari

Luigi Scarso