

SUMMARY OF PDF EXPORT PROJECT

Scribus GSoC '09

Thach Tran

August 24, 2009

1 Recap

This project was formed to extend the PDF export capability of Scribus. It basically covers two tasks.

- Supporting PDF/X-1a and PDF/X-4 export.
- Extending the embedding PDFs feature of Scribus to make it more or less “profile/standard aware”. See the project proposal on Scribus’s wiki for more details.

2 PDF/X-1a and PDF/X-4

Most of the work was done in `pdflib_core` to correctly export PDFs that are PDF/X-1a and PDF/X-4 compliant. Based on the existing code for exporting PDF/X-3, PDF/X-1a is just further forbidding the uses of RGB and ICC-based colors. On the contrary, PDF/X-4 is based on PDF/X-3 but further allows the uses of transparency and live layers. This is also quite simple to implement since `pdflib_core` is already capable of optionally exporting transparency (PDF 1.4 compliant) and live layers (PDF 1.5 compliant).

A little complication was encountered when PDF/X-4 asked for a XMP metadata stream to be included. Exiv2 is a metadata handling library which was proved to be suitable for this task (a small experiment was run). However, including another dependency for Scribus is not so much preferred at the time and including the whole Exiv2 code into Scribus just to handle a small piece of XMP is a bit overkill. So, the XMP was generated manually using the basic XML support of Qt. This is done in `PDFLibCore::generateXMP`. Taking a look at the code, one could quickly see it’s a very low level XML

generating and does not exhibit very clear the semantics of XMP (and consequently hard to maintain). For future work, I certainly wish a piece of XMP will be included in every PDF that Scribus generates as it’s quite a standard nowadays. Then, it would be quite convincing to use Exiv2 as another dependency and get rid of the ugly, low-level code as of now. As a side note, the info in XMP should be exactly the same as the Info dictionary included in the PDF’s trailer which `pdflib_core` has been generating.

Next, preflight test for PDF/X-4 compliant in Adobe Acrobat showed that PDF/X-4 documents generated by Scribus failed to follow the encoding rules for non-symbolic TrueType fonts. This is because Scribus uses **Encoding** dictionary with custom **Differences** array to include any fonts. The advantages of this approach are it will not be depended on any predefined encoding schemes, text showing will work simply by specifying the character codes, and it will be able to utilize all characters that the font has. This works great for other font types but, as noted in PDF spec, is not recommended for TrueType. And in fact, PDF/X-4 has forbidden the use of Encoding dictionary with custom **Differences** array for non-symbolic TrueType. At first attempt, a complete rework specifically for TrueType was considered to get around this issue. However, it was later found out that it’s a lot simpler to include TrueType as a CID-keyed font (**Type0** in PDF terms). This approach requires the minimal changes in Scribus’s font handling and also overcomes the 255 characters limit of simple font dictionary (where a font was split into multiple sets of 255 characters in Scribus to include the whole fonts). Generating a **Type0** font dictionary for a TrueType font is quite straightforward. Regarding the encoding, the **Encoding** entry

is set to **Identity-H** in the font dictionary and **CID-ToGIDMap** entry is set to **Identity** in the **Descendant-Font** array. This will make sure text showing by character code currently implemented in `pdflib_core` will work without any major modification. It's also worth noticing that text showing for this type of font will now have to be represented as 4-byte character code for each character.

Finally, the Preflight Verifier needs to be updated to be able to check the Scribus doc against PDF/X-1a and PDF/X-4 profile. One additional property was added to be checked by the PDF/X-1a profile i.e., `bool checkNotCMYKOrSpot` in `checkerPrefs` struct in `prefsstructs.h`. The implementation for that check was actually implemented in `documentchecker.cpp`. Basically, it just checks the fill and line color of the current item to see if there's RGB color; if so, the corresponding error is flagged. The rest of the properties to be checked by PDF/X-1a profile is just the same as in PDF/X-3 profile. For PDF/X-4 profile, it's just the same as PDF/X-3 profile except it doesn't check for transparency. `checkDocument.h` and `checkDocument.cpp` were also extended to correctly display the new type of error introduced in PDF/X-1a profile (color is not CMYK or spot). Lastly, Scribus's preferences was extended to keep the info of two new checker profiles.

3 Extending embedded PDFs feature

The basic idea is Scribus can embedded existing PDFs (which were provided in image frames) while exporting to PDF in the form of PDF's Form **XObject**. This process is pretty much done by blindly copying all the PDF's object over with very small tweak to make the page stream content becomes Form **XObject** in the new file. Thus, in case the target profile has certain requirements, the embedding PDF might not satisfy this, and consequently, including this content will break the final PDF. This part of the project will try to inform the users as much as possible about the mismatches between the embedded PDFs and the target profile.

At the first step, a new source file was added (`pdf_analyzer`) which contains the code to parse and record various properties of a pdf's page content

stream (e.g. color spaces, transparency, fonts, images, etc). `PoDoFo::PdfContentsTokenizer` was the main tool for this task. The `documentchecker` will use this new class to report any issues it can find regarding the incompatibilities between the embedded PDFs and the target check profile.

About the `pdf_analyzer`, it first of all contained a few basic enums and structs for the purposes of reporting various graphic contents of the PDF's page (e.g. `PDFColorSpace`, `PDFFontType`, `PDFFont`, `PDFImage`, etc). The `PDFAnalyzer` object is constructed only by providing the `QString` representing the path of the PDF file which needs analyzing. There's only one public method to be used which is `inspectPDF` which takes the page number to be inspect (since it's just natural for Scribus to embed only one page in a PDF) and returns the list of used `PDFColorSpace`, list of used `PDFFont`, list of used `PDFImage`, and a boolean indicating there's transparency in the page. See `pdf_analyzer.h` for more info.

The actual code for parsing and recording all these graphic contents was implemented in `inspectCanvas` where the parameters passed to this routine are the same as `inspectPDF` above except instead of taking the page number this routine takes a `PoDoFo::PdfCanvas` pointer indicating exactly the content needs to be inspected. Since we could be dealing with either a page content stream (`PoDoFo::PdfPage`) or content in a form **XObject** (`PoDoFo::PdfXObject`) which is referenced from the page content, this method is general enough (`PdfPage` and `PdfXObject` are subclasses of `PdfCanvas`) to be used in both cases and also very convenient to be used recursively.

The code to parse and record all the graphic content is nothing tricky but requires quite a bit of extra works to be able to cover the syntax complexity of PDF. Below is a few things worth noticing

- Color space of the graphic state could be altered by: **g**, **G**, **k**, **K**, **rg**, **RG**, **cs**, and **CS** operator, **ColorSpace** entry in image dictionary (both image **XObject** and inline image). There's always two color spaces that are currently in use, one for stroking and one for filling.
- The current font could be altered by **Tf** operator and the generic graphic state operator **gs** (where the supplied graphic state parameter

dictionary contains a **Font** entry).

- Image in PDF can be image **XObject** of which the image stream is defined in the current resources dictionary or the image could be inlined in the content stream using **BI**, **ID**, **EI** operators (rarely used nowadays).
- There's a few things in a PDF can indicate the uses of transparency in the document.
 - The transparency group in the **Page/XObject** dictionary.
 - Graphic state parameter dictionaries which are applied by **gs** operator containing **BM** entry (blend mode) which is other than **Normal** or **Compatible**; **ca/CA** (alpha constant) entry which is smaller than 1; and **SMask** entry (softmask) which is not **None**.

In order to report the true dpi of images in the PDF, the current transformation matrix (CTM) of the page needs to be maintained. Since this is one of the graphic state parameters, it could be a good idea to keep track of the current graphic state as a whole. The **PDFGraphicState** struct was introduced to achieve this. The code has started updating the current graphic state as it moves along the content stream. Only the CTM is used to identify the true dpi of images, the rest might be useful in the future. See the code for more details on what properties are maintained in the current graphic state.

Regarding the actual checking of the embedded PDFs against various target profiles. A few new errors were introduced to cover some new issues.

- In case of PDF/X-3 and PDF/X-4, the output intend is one of the requirements where the color profile of the target printer is included. The *device* color spaces used in the document has to be consistent with this printer color profile. In Scribus, the printer profile for a document can be set in the CMS preferences. Within the context of checking embedded PDF, we could use this profile to check if the device color spaces used in the embedded PDFs match with the output intend. For example, if the printer profile is a CMYK profile and the embedded PDF contains **DeviceRGB** or **DeviceGray**, an error should be reported.

- Regarding the font used in the embedded PDFs, two new errors could be reported are: not all fonts are embedded in the file and the embedded font is OpenType (which was only OK in PDF/X-4 since it's based on PDF 1.6).
- Other errors could be reported in the embedded PDFs but doesn't need new error type are:
 - Color is not CMYK or spot (PDF/X-1a)
 - There's transparency (only OK in PDF 1.4, PDF 1.5 and PDF/X-4)
 - DPI of image is too low or too high

That's all I was able to do given the time frame of GSoC and it by no mean can cover all the things there is to embedded PDF in Scribus. We could certainly add things as we move along and discover new features; as well as listen to feedbacks from the community. One thing could easily use some more work is the color reporting; the device color spaces and CIE-based color spaces are fine as far as I can see but the special color spaces such as **Pattern**, **Separation**, **DeviceN** will need further investigation in the code to do a more thorough analysis.