An Introduction to PDF with CamlPDF

John Whitington

March 8, 2010

Coherent Graphics Ltd

For bug reports, feature requests and comments, email
contact@coherentgraphics.co.uk

©2010 Coherent Graphics Limited. All rights reserved.

Adobe, Acrobat, Adobe PDF, Adobe Reader and PostScript are registered trademarks of Adobe Systems Incorporated.

A First Program

To build CamlPDF, navigate to the source directory and type

```
make
```

If this fails, follow the full instructions in the file 'INSTALL'. To build the examples, type

```
make -f examplesmake
```

Now run a simple example to build the file `hello.pdf`

```
./pdfhello
```

Now build an interactive top level, which we will use to explore CamlPDF

```
make top
```

Now we can enter the top level:

```
./camlpdf.top
        Objective Caml version 3.11.0

#
```

The Pdfread module allows us to load PDF files into memory. The raw PDF data is parsed into a structured OCaml value of type Pdf.pdfdoc:

```
# let pdf = Pdfread.pdf_of_file None "hello.pdf";;
val pdf : Pdf.pdfdoc =
  {Pdf.major = 1;
   Pdf.minor = 0;
   Pdf.root = 4;
   Pdf.objects = <abstr>;
   Pdf.trailerdict =
    Pdf.Dictionary
     [("/Size", Pdf.Integer 4);
      ("/Root", Pdf.Indirect 4);
      ("/ID",
         Pdf.Array
          [Pdf.String "?b??\155\144?r\155l?vp\"\014?";
           Pdf.String "?b??\155\144?r\155l?vp\"\014?"])]}
```

Looking at the parts of the Pdf.pdfdoc record type:

- Pdf.major and Pdf.minor - the parts of the PDF version number. Here, PDF Version 1.0
- Pdf.root - the object number of the 'root object' of the PDF (A PDF is a directed graph of objects, indexed by number)
- Pdf.objects - the PDF objects
- Pdf.trailerdict - the trailer dictionary. This is a distinguished PDF object containing a number of commonly used per-file items. Pdf.trailerdict has type Pdf.pdfobject, which represents all possible PDF data.

Diversion: A Look at hello.pdf

Here is the contents of the file `hello.pdf`, as you might see it in a text editor, annotated with some explanatory comments.

```
%PDF-1.0 Header
%%$^@
1 0 obj Object 1...
<< /Type /Pages /Kids [ 3 0 R ] /Count 1 >> ...which is the catalogue of pages
endobj
2 0 obj This object is a stream, which is a dictionary plus some binary data
<< /Length 102 >>
stream Usually compressed, but plain here for ease of reading
1.000000 0.000000 0.000000 1.000000 50.000000 770.000000 cm
BT /F0 36.000000 Tf (Hello, World!) Tj ET The page content, a bit like PostScript
endstream
endobj
3 0 obj The page object,
<< /Type /Page
   /Parent 1 0 R The syntax "1 0 R" means a reference to Object 1
   /Resources
      << /Font The font dictionary
            << /F0
              << /Type /Font /Subtype /Type1 /BaseFont /Times-Italic >>
            >> >>
   /MediaBox [ 0.000000 0.000000 595.275591 841.889764 ] The page dimensions
   /Rotate 0
   /Contents [ 2 0 R ] >> Reference to contents in object 2
endobj
4 0 obj
<< /Type /Catalog /Pages 1 0 R >> The root object
endobj
xref The cross-reference table, listing the byte offsets of each object for random access.
0 5
0000000000 65535 f
0000000015 00000 n
0000000074 00000 n
0000000227 00000 n
0000000449 00000 n
```

```
trailer The trailer dictionary
<< /Size 5 /Root 4 0 R /ID [ (:.]ZŁzE) (:.?ZŁzE) ] >>
startxref
498 The trailer
%%EOF
```

Saving the Document

The Pdf.pdfdoc data type is a record of mutable values. Let's change the PDF Version number and write the file.

```
# pdf.Pdf.minor <- 1;;
- : unit = ()
# Pdfwrite.pdf_to_file pdf "hello2.pdf";;
- : unit = ()
```

Next Steps

The objects in a PDF document are of type Pdf.pdfobject:

```
type stream = Stream data. Either in memory or still in the file
  | Got of Utility.bytestream
  | ToGet of Pdfio.input * int64 * int64 input, offset, length

type pdfobject =
  | Null
  | Boolean of bool
  | Integer of int
  | Real of float
  | String of string
  | Name of string
  | Array of pdfobject list
  | Dictionary of (string * pdfobject) list
  | Stream of (pdfobject * stream) ref Stream data (see above)
  | Indirect of int A reference to another object
```

For instance the PDF object in the file:

```
3 0 obj
<< /Type /Page
```

```
   /Parent 1 0 R
   /MediaBox [ 0.000000 0.000000 595.275591 841.889764 ]
   /Rotate 0
   /Contents [ 2 0 R ]
>>
end
```

is represented as object number 3 with the Pdf.pdfdoc instance:

```
Dictionary
  ["/Type", Name "/Page";
   "/Parent", Indirect 1;
   "/MediaBox",
       Array [Real 0.; Real 0.; Real 595.275591; Real 841.889764];
   "/Rotate", Integer 0;
   "/Contents", Array [Indirect 2]]
```

Working with Pages

Introduce a command to show the current document, using whatever command opens (or updates) a PDF view on your system:

```
let show pdf =
  Pdfwrite.pdf_to_file pdf "temp.pdf";
  ignore (Sys.command "open temp.pdf");; Customize here
```

The Pdfdoc module deals with PDF pages. We can get the list of pages from a document:

```
# let pages = Pdfdoc.pages_of_pagetree pdf;;
val pages : Pdfdoc.page list =
  [{Pdfdoc.content =
     [Pdf.Stream
       {contents =
         (Pdf.Dictionary
           [("/Length", Pdf.Integer 102)], Pdf.Got <abstr>)}];
    Pdfdoc.mediabox =
     Pdf.Array
      [Pdf.Integer 0; Pdf.Integer 0; Pdf.Real 595.275591;
       Pdf.Real 841.889764];
    Pdfdoc.resources =
     Pdf.Dictionary
      [("/Font",
```

```
        Pdf.Dictionary
         [("/F0",
           Pdf.Dictionary
            [("/Type", Pdf.Name "/Font");
             ("/Subtype", Pdf.Name "/Type1");
             ("/BaseFont", Pdf.Name "/Times-Italic")])];
    Pdfdoc.rotate = Pdfdoc.Rotate0;
    Pdfdoc.rest = Pdf.Dictionary []}]
```

Each page is a record containing five things:

- Pdfdoc.content An ordered list of pdf objects representing the one or more streams containing the graphical data for the page.
- Pdfdoc.mediabox The page dimensions
- Pdfdoc.resources The resources dictionary for a page, which contains the fonts, colour spaces and so on for the page.
- Pdfdoc.rotate The viewing rotation for the page.
- Pdfdoc.rest The rest of the page dictionary (i.e that which has not been separated into the items above).

Let's change the viewing rotation to 90 degrees:

```
# let page = {(List.hd pages) with Pdfdoc.rotate = Pdfdoc.Rotate90};;
val page : Pdfdoc.page = ...
# let pdf = Pdfdoc.change_pages false pdf [page];;
val pdf : Pdf.pdfdoc = ...
# show pdf;;
- : unit
```

Now change the rotation back: we're going to work with graphics next, and the viewing roation would confuse:

```
# let page = List.hd pages;;
val page : Pdfdoc.page = ...
# let pdf = Pdfdoc.change_pages false pdf [page];;
val pdf : Pdf.pdfdoc = ...
# show pdf;;
- : unit
```

Graphics and Text

The Pdfpages module represents the graphical content of each page, which is formed of PostScript-like operators which draw the page. Let's get the operator list from the page:

```
# let ops =
    Pdfpages.parse_operators
      pdf page.Pdfdoc.resources page.Pdfdoc.content;;
val ops : Pdfpages.operator list =
  [Pdfpages.Op_cm
    {Transform.a = 1.; Transform.b = 0.;
     Transform.c = 0.; Transform.d = 1.;
     Transform.e = 50.; Transform.f = 770.};
   Pdfpages.Op_BT;
   Pdfpages.Op_Tf ("/F0", 36.);
   Pdfpages.Op_Tj "Hello, World!";
   Pdfpages.Op_ET]
```

The Op_cm operator alters the graphics matrix to position the text. Op_BT and Op_ET mark the beginning and end of a text section. Op_Tf chooses 36pt Times Italic (which is font F0 in the page's font dictionary in its resources) and Op_Tj paints the text.

Let's add operators to underline the text– Op_m to move, Op_l to draw a line and Op_S to stroke the path. We calculate the width of the underline using the Fonttables module to get the raw width of the string in millipoints, adjusting for font size and converting to points.

```
# let width = Fonttables.textwidth Pdftext.TimesItalic "Hello, World!";;
val width : int = 5423

# let actual_width = float width *. 36. /. 1000.;;
val actual_width : float = 195.228

# let ops' =
  ops @
    [Pdfpages.Op_m (0., 0.);
     Pdfpages.Op_l (actual_width, 0.);
     Pdfpages.Op_S];;
val ops' : Pdfpages.op list = ...
```

and make the new content stream:

```
# let stream = Pdfpages.stream_of_ops ops';;
val stream : Pdf.pdfobject =
  Pdf.Stream
   {contents =
```

```
      (Pdf.Dictionary [("/Length", Pdf.Integer 146)], Pdf.Got <abstr>)}
```

and add it to the page, and replace the page in the PDF.

```
# let page' = {page with Pdfdoc.content = [stream]};;
val page' : Pdfdoc.page = ...

# let pdf = Pdfdoc.change_pages false pdf [page'];;
val pdf : Pdf.pdfdoc = ...
```

and show it:

```
# show pdf;;
- : unit ()
```

Next Steps

Summary of CamlPDF modules:

Module	Description	Status
Utility	General functions	Complete
Pdfio	Generic Input/Output	Complete
Transform	Affine transforms in two dimensions	Complete
Units	Units and unit conversion	Complete
Paper	Media sizes	Complete
Pdf	Representing PDF files in memory	Complete
Pdfcrypt	Encryption and Decryption	Complete
Pdfwrite	Writing PDF files	Complete
Pdfcodec	Encoding and Decoding PDF streams	Complete
Pdfread	Reading PDF files	Complete
Pdfpages	Parsing PDF graphics streams	Complete
Pdfdoc	Document-level functions	Complete
Pdfannot	Annotations	Complete
Pdffun	Parsing and evaluating PDF functions	Some types of function missing
Pdfspace	Colour spaces	Some types of colourspace missing
Pdfimage	Extract and create images	Incomplete and unsupported
Glyphlist	Glyph lists	Complete
Pdftext	Parsing fonts and extracting text	Text extraction incomplete
Fonttables	Standard PDF fonts	Complete
Pdfgraphics	Structured graphics	Incomplete and unsupported
Pdfshapes	Basic shapes	Complete
Pdfmarks	Bookmarks	Incomplete and unsupported
Pdfdate	Representing and parsing PDF dates	Complete
Cff	Convert a CFF Type 1 font to a Type 3 font	Incomplete and unsupported

To make the HTML documentation for CamlPDF, type `make documents`.

Examples shipped with CamlPDF:

`pdfhello.ml`	Build a "Hello, World!" PDF from scratch
`pdfdecomp.ml`	Command line utility to decompress a PDF
`pdfmerge.ml`	Command line utility to merge PDF files
`pdfdraft.ml`	Command line utility to make draft documents
`pdftest.ml`	Reads and interprets a file to test CamlPDF's major functionality
`pdfdecrypt.ml`	Command line utility to decrypt a PDF file

Further Reading

For any serious work, you will need the PDF Reference Manual
`http://www.adobe.com/devnet/acrobat/pdfs/PDF32000_2008.pdf`
(also available in print form).