

DAVID BLANK-EDELMAN

## practical Perl tools: Peter Piper picked a peck of PDFs



David N. Blank-Edelman is the Director of Technology at the Northeastern University College of Computer and Information Science and the author of the O'Reilly book *Perl for System Administration*. He has spent the last 20+ years as a system/network administrator in large multi-platform environments, including Brandeis University, Cambridge Technology Group, and the MIT Media Laboratory. He was the program chair of the LISA 2005 conference and one of the LISA 2006 Invited Talks co-chairs.

*dnb@ccs.neu.edu*

### THE ADOBE PORTABLE DOCUMENT

Format (PDF) has become a lingua franca in the business and technology world. I'd hazard a guess that you probably read and perhaps generate one or several PDF documents a day as part of your daily routine. Documentation, invoices, electronic books, copies of presentations, and a whole bunch of other document types now commonly live in PDF format. Even though I write this column in plain text, when it gets typeset for this publication, a draft proof copy comes back to me in PDF format. I just ran a quick check, and I find that I have 2,678 PDF files on the laptop being used to write this column. That laptop (a Mac) treats PDF files as a "native" format and knows how to create and read them right out of the box.

I don't mind swimming in documents in this format because it is an "open standard." Adobe distributes the specifications for how PDF documents are constructed. Everyone is free to create programs that read and write this format, with Adobe's royalty-free blessing. Adobe announced in January of this year that they plan to submit the latest version of the PDF spec (1.7) to the International Organization for Standardization (ISO) so it can become a standard standard (vs. a non-standardized standard, I suppose. Paging Nick Stoughton . . .).

The PDF format may be open and ubiquitous, but I suspect I'm not alone in thinking about PDF files as a kind of black-box magic. My PDF-generating applications create PDF files, my PDF-reading applications display their contents, and that's close to the level of expertise I've desired on the subject. It is a little like the PostScript language. I've had to suss out enough PostScript to hand-edit recalcitrant PostScript files less than ten times in my life, and I didn't enjoy the process. (Remind me to tell you some day about the person I met who credibly claimed to have written an entire Web server in PostScript.)

With that level of technical apathy in mind, it will make sense that this is a column about creating and manipulating PDF files from Perl using higher-level interfaces. If you want to forge individual PDF XObjects yourself you'll need to use different Perl modules from those discussed here. If you want to know how to work with PDF files without

knowing too much about just how they represent their data, you've come to the right place.

---

## Generating PDF Files from Perl

---

Let's start with nothing and see if we can wind up with something. There are a number of modules at the right level of abstraction for our purposes that can create new PDF files. Two of the popular packages are PDF::API2 and the Perl bindings to the commercial (with a more limited free version) PDFlib package. We'll take a really quick look at how to use both of them, starting with the free package.

PDF::API2 has an extensive list of PDF features, such as support for different font types and graphic formats. Unfortunately, this power comes with a little more pain than I'd prefer. The documentation assumes you already have some PDF experience and you are just searching for the module's methods to make use of your experience. A simple "Hello World!" looks like this (from the doc):

```
use PDF::API2;

$pdf = PDF::API2->new;

$fnt = $pdf->corefont('Helvetica-Bold');

$page = $pdf->page;
$page->mediabox('A4');

$gfx = $page->gfx;
$gfx->textlabel(200,700,$fnt,20,'Hello World !');

$pdf->saveas('/this/new/document.pdf');
$pdf->end;
```

Let's walk through this example line by line. After loading the module and creating a new PDF::API2 module, the first step is to request a font object. Think of it as a pointer to the font we will use later when we draw text. It is called a "core font" because the PDF standard blesses 14 fonts as "core fonts"; these are always available on any system. With this in place, we create a page object and set the MediaBox (i.e., the physical size) of that page. We then ask for a handle into the graphics content object of that page. Using this object, we can finally write some text onto the page. The text is placed at coordinates 200,700 (using the wacky PDF system of 0,0 being in the lower left of the page) and is rendered at a size of 20 points. The last two statements save the data out to the file and destroy the PDF object.

Whew . . . and that's a simple example. Just figuring out this little snippet of code can run you ragged if you are not familiar with the standard PDF nomenclature. For example, when I was first trying to understand what `$page->gfx` did I found I had to consult the source code in three separate submodules just to get the basic what, why, and wherefore for that line of code. The documentation is equally terse on other matters; for example, the `textlabel()` documentation lists its arguments, but it never says what the units for size should be (for that, I had to go track down the official PDF specification at [http://www.adobe.com/devnet/pdf/pdf\\_reference.html](http://www.adobe.com/devnet/pdf/pdf_reference.html)). I'm not complaining as much as I'm warning you that you may be in for a bumpy ride with this module.

I'm not the only person who has noticed these shortcomings. There are several helper modules that provide a less daunting face for PDF::API2. For example, PDF::API2::Simple lets you write code like this:

```

use PDF::API2::Simple;
my $pdf = PDF::API2::Simple->new( file => 'output.pdf' );
$pdf->add_font("Helvetica-Bold");      # load the font
$pdf->add_page();                       # start a new page
$pdf->text('Hello World!',
          x => 200, y => 700,
          font => 'Helvetica-Bold', font_size => 20);
$pdf->save();

```

The other approach I'd recommend exploring when it comes to PDF creation is the use of the commercial package by the German company PDFlib GmbH ([www.pdflib.com](http://www.pdflib.com)). PDFlib GmbH produces an exceptionally full-featured library for PDF creation and modification, with bindings for many different languages: Cobol, COM, C, C++, Java, .NET, Perl, PHP, Python, REALbasic, RPG, Ruby, and Tcl. Its library can basically handle anything you'd want to do relating to PDF files, including functions far beyond what PDF::API2 can handle. If you need to do heavy-duty PDF production programmatically, this is going to be a good bet.

PDFlib GmbH also provides a "Lite" version of their product that is free for noncommercial, personal, open source developer and research use. It is a considerably smaller subset of the commercial offerings, but it probably can do most of what the casual user needs. Let's take a quick peek at how to use PDFlib Lite from Perl.

There are two interfaces for this package we could consider using: the one that ships with PDFlib Lite and a wrapper module for it called PDFLib, which provides an object-oriented interface to it. The PDFLib wrapper module was last updated three years ago, so we're going to stick to the bundled version for this example. To help continue the comparison we've already started, let's look at a simple "Hello World!" example using this module as well (adapted from the example in the PDFlib Lite distribution):

```

use pdflib_pl;
my $pdf = PDF_new();

# ask each function to return -1 if there is an error
PDF_set_parameter( $pdf, "errorpolicy", "return" );

if ( PDF_begin_document( $pdf, 'hello.pdf', '' ) == -1 ) {
    die 'Unable to begin document: ' . PDF_get_errmsg($pdf) . "\n";
}

# 612 x 792 points is US letter-sized paper
PDF_begin_page_ext( $pdf, 612, 792, '' );

# load the font (in a particular encoding)
my $font = PDF_load_font( $pdf, "Helvetica-Bold", "winansi", "" );
if ( $font == -1 ) {
    die 'Unable to load font: ' . PDF_get_errmsg($pdf) . "\n";
}

# make it the current font
PDF_setfont( $pdf, $font, 20.0 );

# place and print the text on the page
PDF_set_text_pos( $pdf, 200, 700 );
PDF_show( $pdf, ' Hello World !' );

# finish the page
PDF_end_page_ext( $pdf, '' );

# finish the document

```

```
PDF_end_document( $pdf, '' );  
  
# be nice and destroy the pdf object  
PDF_delete($pdf);
```

I don't want to bore you with any more "Hello World" programs. We've seen the very basics of creating PDFs from scratch. We can get more complicated by importing images, drawing lines and shapes, and messing with text formatting and placement in a fairly straightforward way. Rather than going deeper into PDF creation, I want to switch topics now so we have enough space to cover the second activity people would like to use Perl for when dealing with PDFs.

---

## Manipulating Existing PDF Files with Perl

---

Even if you don't need to create your own custom PDF files programmatically, you probably occasionally need to modify and manipulate existing files. For example, if you need to send someone the answer to a question found buried deep in the documentation, it may be better to send them just a few pages rather than the whole 800-page manual. Going in the opposite direction, you may want to concatenate several separate documents so you can send them as a single file to avoid confusion. It could be handy to extract all of the images or text from a PDF file to separate files. Perhaps you'd like to add a footer on all of the pages in an existing document with a message such as "Highly Confidential—Eat if Captured." All of these things and more are available to you courtesy of the right Perl modules.

Did I say "modules"? You could use separate modules (including one of the commercial PDFlib offerings) but there's actually an all-singing, all-dancing PDF manipulation module called `CAM::PDF` that can handle all of these tasks for you. Let's look at how to perform some of the tasks just mentioned using it. Before we go on, let me slake your curiosity by saying that the `CAM::` in `CAM::PDF` comes from "Clotho Advanced Media," the company that originally developed the module.

Starting at the top of our wish list, to extract pages 1, 3, and 12 from a PDF file, we could use something like this:

```
use CAM::PDF;  
  
my $pdf = CAM::PDF->new('pdf_reference.pdf');  
$pdf->extractPages( 1, 3, 12 );  
$pdf->cleanoutput('output.pdf');
```

Yes, it is that easy. We create an object that points to the input file, tell it to extract the right pages, and then write the document to a new file with `cleanoutput()` (as opposed to `save()`, which will append to the original file).

Appending two files is similarly easy:

```
use CAM::PDF;  
  
my $pdf1 = CAM::PDF->new('pdf1.pdf');  
my $pdf2 = CAM::PDF->new('pdf2.pdf');  
$pdf1->appendPDF($pdf2);  
$pdf1->cleanoutput('concat.pdf');
```

`CAM::PDF` comes with scripts to handle jpeg and text extraction and footer addition ("stamping"), so I won't include that code here. It comes with quite a few utility scripts like this, so it is worth your while to check out the package.

As a final postscript to this section, and as we fade into the sunset, I do want to mention that if CAM::PDF is not your cup of tea, the other module worth your consideration should be PDF::Reuse. PDF::Reuse's whole raison d'être was the desire to take an existing PDF file and use it as a template for the creation of other PDF files. For example, you could take a small PDF file with a picture of a business card and have it create a document with this card repeated in columns on the page for mass printing. Another possibility would be to send someone a customized PDF document with hyperlinks in the body that were personalized for the particular user. You probably can think of other ways this could come in handy. Both CAM::PDF and PDF::Reuse will serve you well in these cases.

I hope this column has demystified the process of PDF file creation and modification just enough so you can get what you want done without having to devote too much of your limited brain space to PDF minutiae. Take care, and I'll see you next time.