A Simple Step-by-Step Reportlab Tutorial

March 8, 2010 Cross-Platform, Python Python PDF Series, Reportlab Mike

The subtitle for this article could easily be "How To Create PDFs with Python", but WordPress doesn't support that. Anyway, the premier PDF library in Python is Reportlab. It is not distributed with that standard library, so you'll need to download it if you want to run the examples in this tutorial. There will also be at least one example of how to put an image into a PDF, which means you'll also need the Pillow package (PIL).

Installation

Reportlab supports most of the normal Python installation methods. For the old **Reportlab 2.x** versions you have the option of downloading the source and running "python setup.py install" or running a binary installer (on Windows).

For the newer **Reportlab 3.x**, you can now use pip on all platforms:

```
pip install reportlab
```

Note that Reportlab 3.x only supports **Python 2.7** and **Python 3.3+**. If you are on an older version of Python 2, then you have to use Reportlab 2.x.

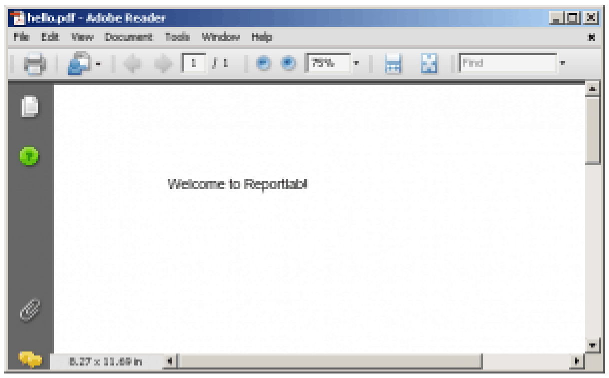
Creating a Simple PDF

Reportlab has decent documentation. What I mean by that is that the documentation gives you just enough to get started, but when you find something slightly complex to do, you get to figure it out on your own. Just recently, they added a Code Snippets section to their website that will hopefully become a recipe book of cool tips and tricks and also help ameliorate this issue. But enough about that. Let's see how to actually create something!

In Reportlab, the lowest-level component that's used regularly is the *canvas* object from the *pdfgen* package. The functions in this package allow you to "paint" a document with your text, images, lines or whatever. I've heard some people describe this as writing in PostScript. I doubt it's really that bad. In my experience, it's actually a lot like using a GUI toolkit to layout widgets in specific locations. Let's see how the canvas object works:

```
from reportlab.pdfgen import canvas

c = canvas.Canvas("hello.pdf")
c.drawString(100,750,"Welcome to Reportlab!")
c.save()
```

You should end up with a PDF that looks something like this:



The first thing to notice about this code is that if we want to save the PDF, we need to supply a file name to the Canvas object. This can be an absolute path or a relative path. In this example, it should create the PDF in the same location that you run the script from. The next piece of the puzzle is the *drawString* method. This will draw text wherever you tell it to. When using the canvas object, it starts at the bottom left of the page, so for this example, we told it to draw the string 100 points from the left margin and 750 points from the bottom of the page (1 point = 1/72 inch). You can change this default in the Canvas constructor by passing a zero to the *bottomup* keyword argument. However, I'm not exactly sure what will happen if you do that as the Reportlab user guide isn't clear on this topic. I think it will change the start point to the top left though. The final piece in the code above is to save your PDF.

That was easy! You've just created a really simple PDF! Note that the default Canvas size is A4, so if you happen to be American, you'll probably want to change that to letter size. This is easy to do in Reportlab. All you need to do is the following:

```
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas
```

```
canvas = canvas.Canvas('myfile.pdf', pagesize=letter)
width, height = letter
```

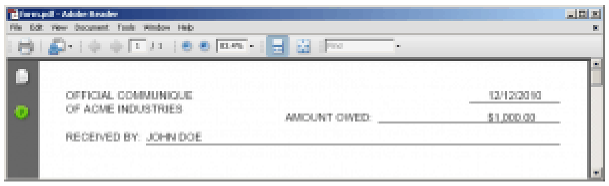
The main reason to grab the width and height is that you can use them for calculations to decide when to add a page break or help define margins. Let's take a quick look at the constructor for the Canvas object to see what other options we have:

```
def __init__(self,filename,
    pagesize=letter,
    bottomup = 1,
    pageCompression=0,
    encoding=rl_config.defaultEncoding,
    verbosity=0
    encrypt=None):
```

The above was pulled directly from the Reportlab User Guide, page 11. You can read about the other options in their guide if you want the full details.

Now let's do something a little more complicated and useful.

A Little Form, Little Function



OFFICIAL COMMUNIQUE
OF ACME INDUSTRIES

12/12/2010

AMOUNT OWED: $1,000.00

RECEIVED BY: JOHN DOE

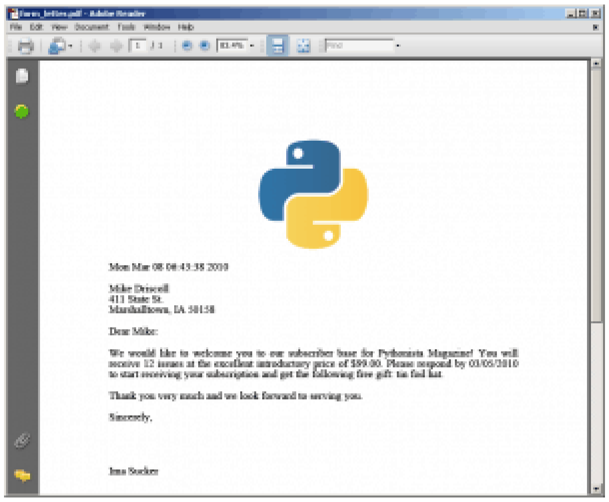
In this example, we'll create a partial printable form. As far as I can tell, Reportlab doesn't support the fillable forms that were added to Adobe products a few years ago. Anyway, let's take a look at some code!

```
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas

canvas = canvas.Canvas("form.pdf", pagesize=letter)
canvas.setLineWidth(.3)
canvas.setFont('Helvetica', 12)

canvas.drawString(30,750,'OFFICIAL COMMUNIQUE')
canvas.drawString(30,735,'OF ACME INDUSTRIES')
canvas.drawString(500,750,"12/12/2010")
canvas.line(480,747,580,747)

canvas.drawString(275,725,'AMOUNT OWED:')
canvas.drawString(500,725,"$1,000.00")
canvas.line(378,723,580,723)

canvas.drawString(30,703,'RECEIVED BY:')
canvas.line(120,700,580,700)
canvas.drawString(120,703,"JOHN DOE")

canvas.save()
```

This is based on an actual receipt I created at work. The main difference between this one and the previous example is the *canvas.line* code. You can use it to draw lines on your documents by passing two X/Y pairs. I've used this functionality to create grids, although it's pretty tedious. Other points of interest in this code include the setLineWidth(.3) command, which tells Reportlab how thick or thin the line should be; and the setFont('Helvetica', 12) command, which allows us to specify a specific font and point size.

Our next example will build on what we've learned so far, but also introduce us to the concept of "flowables".

Going with the Flow


form_letters.pdf - Adobe Reader
File Edit View Document Tools Window Help

Mon Mar 08 08:43:38 2010

Mike Driscoll
411 State St.
Marshalltown, IA 50158

Dear Mike:

We would like to welcome you to our subscriber base for Pythonista Magazine! You will receive 12 issues at the excellent introductory price of $99.00. Please respond by 03/05/2010 to start receiving your subscription and get the following free gift: tin foil hat.

Thank you very much and we look forward to serving you.

Sincerely,

Ima Sucker

If you're in advertising or do any kind of work with form letters, then Reportlab makes for an excellent addition to your arsenal. We use it to create form letters for people who have overdue parking tickets. The following example is based on some code I wrote for that application, although the letter is quite a bit different. (Note that the code below will not run without the Python Imaging Library)

```
import time
from reportlab.lib.enums import TA_JUSTIFY
from reportlab.lib.pagesizes import letter
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer,
Image
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
from reportlab.lib.units import inch

doc = SimpleDocTemplate("form_letter.pdf",pagesize=letter,
                        rightMargin=72,leftMargin=72,
                        topMargin=72,bottomMargin=18)
Story=[]
logo = "python_logo.png"
magName = "Pythonista"
issueNum = 12
subPrice = "99.00"
limitedDate = "03/05/2010"
freeGift = "tin foil hat"

formatted_time = time.ctime()
full_name = "Mike Driscoll"
address_parts = ["411 State St.", "Marshalltown, IA 50158"]

im = Image(logo, 2*inch, 2*inch)
Story.append(im)

styles=getSampleStyleSheet()
styles.add(ParagraphStyle(name='Justify', alignment=TA_JUSTIFY))
ptext = '<font size=12>%s</font>' % formatted_time

Story.append(Paragraph(ptext, styles["Normal"]))
Story.append(Spacer(1, 12))

# Create return address
ptext = '<font size=12>%s</font>' % full_name
Story.append(Paragraph(ptext, styles["Normal"]))
for part in address_parts:
    ptext = '<font size=12>%s</font>' % part.strip()
    Story.append(Paragraph(ptext, styles["Normal"]))

Story.append(Spacer(1, 12))
ptext = '<font size=12>Dear %s:</font>' % full_name.split()[0].strip()
Story.append(Paragraph(ptext, styles["Normal"]))
Story.append(Spacer(1, 12))

ptext = '<font size=12>We would like to welcome you to our subscriber
base for %s Magazine! \
        You will receive %s issues at the excellent introductory price
of $%s. Please respond by\
        %s to start receiving your subscription and get the following
free gift: %s.</font>' % (magName,
issueNum,

subPrice,

limitedDate,

freeGift)
Story.append(Paragraph(ptext, styles["Justify"]))
Story.append(Spacer(1, 12))


ptext = '<font size=12>Thank you very much and we look forward to
serving you.</font>'
Story.append(Paragraph(ptext, styles["Justify"]))
Story.append(Spacer(1, 12))
ptext = '<font size=12>Sincerely,</font>'
Story.append(Paragraph(ptext, styles["Normal"]))
Story.append(Spacer(1, 48))
ptext = '<font size=12>Ima Sucker</font>'
Story.append(Paragraph(ptext, styles["Normal"]))
Story.append(Spacer(1, 12))
doc.build(Story)
```

Well, that's a lot more code than our previous examples contained. We'll need to look over it slowly to understand everything that's going on. When you're ready, just continue reading.

The first part that we need to look at are the new imports:

```
from reportlab.lib.enums import TA_JUSTIFY
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer,
Image
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
from reportlab.lib.units import inch
```

From enums, we import "TA_JUSTIFY", which allows our strings to have the *justified* format. There are a number of other constants we could import that would allow us to right or left justify our text and do other fun things. Next is the platypus (which stands for Page Layout and Typography Using Scripts) module. It contains lots of modules, but probably the most important of them are the flowables, such as Paragraph. A flowable typically has the following abilities: *wrap*, *draw* and sometimes *split*. They are used to make writing paragraphs, tables and other constructs over multiple pages easier to do.

The SimpleDocTemplate class allows us to set up margins, page size, filename and a bunch of other settings for our document all in one place. A Spacer is good for adding a line of blank space, like a paragraph break. The Image class utilizes the Python Image Library to allow easy insertion and manipulation of images in your PDF.

The getSampleStyleSheet gets a set of default styles that we can use in our PDF. ParagraphStyle is used to set our paragraph's text alignment in this example, but it can do much more than that (see page 67 of the user guide). Finally, the *inch* is a unit of measurement to help in positioning items on your PDF. You can see this in action where we position the logo: Image(logo, 2*inch, 2*inch). This means that the logo will be two inches from the top and two inches from the left.

I don't recall the reason why Reportlab's examples use a Story list, but that's how we'll do it here as well. Basically you create a line of text, a table, and image or whatever and append it to the Story list. You'll see that throughout the entire example. The first time we use it is when we add the image. Before we look at the next instance, we'll need to look at how we add a style to our styles object:

```
styles.add(ParagraphStyle(name='Justify', alignment=TA_JUSTIFY))
```

The reason this is important is because you can use the style list to apply various paragraph alignment settings (and more) to text in your document. In the code above, we create a ParagraphStyle called "Justify". All it does is justify our text. You'll see an example of this later in the text. For now, let's look at a quick example:

```
ptext = '<font size=12>%s</font>' % formatted_time
Story.append(Paragraph(ptext, styles["Normal"]))
```

For our first line of text, we use the Paragraph class. As you can see, the Paragraph class accepts some HTML-like tags. In this instance, we set the font's point size to 12 and use the normal style (which is left aligned, among other things). The rest of the example is pretty much the same, just with Spacers thrown in here and there. At the end, we call *doc.build* to create the document.

Wrapping Up

Now you know the basics for creating PDFs in Python using Reportlab. We didn't even scratch the surface of what all you can do with Reportlab though. Some examples include tables, graphs, paginating, color overprinting, hyperlinks, graphics and much more. I highly recommend that you download the module along with its user guide and give it a try!

Further Reading

- Reportlab Home Page
- Generating Reports with Charts
- A series on Reportlab

- Getting Started with Reportlab