
# Introduction

## Creating PDF reports using Python and ReportLab

I'll be covering the use of Python and the ReportLab library to create quality PDF documents such as database reports. I'll show you how to send these directly to a web client using CGI, and how to pull the data you need out of a PostgreSQL database.


# What is Python?

Python is an interpreted, object oriented language that's useful for beginner and advanced users.

- Useful built-in library
- High level data structures
- Clean syntax
- Very flexible
- Serious language. Metaclasses, multiple inheritance, etc.


# What is ReportLab

ReportLab is an open source PDF creation library for Python.

***Powerful yet simple to use***

- Exposes the low-level PDF "canvas"
- Provides high level document formatting and layout
- Includes support for tables, graphs, etc
- Supports the creation of vector graphics

***Portable***

- Runs on Linux/BSD/UNIX, MacOS X, Windows
- Works under both cPython and Jython.


## What is ReportLab *Not*?

- *WYSWIG* publishing like Quark, InDesign, or Scribus.
- A report building program like Crystal Reports
- A document formatting system like LaTeX (though you can use it for that, too)


# ReportLab Features

## Efficient

PDF has a reputation for being bloated and inefficient. This is at only partly justified.

- Most PDFs are created from output optimised for PostScript printers.
- Lots of redudant or unused information
- Hard to identify and re-use repetitive content.
- PDF creation has a very different set of needs and priorities.
- Distillers have their work cut out for them just handling quirky PostScript from apps.


ReportLab, on the other hand, generates small, efficient, and clean PDFs.

- Outputs images and fonts once, then re-uses them throughout the document.
- Supports PDF forms to allow the efficient re-use of repeated content.
- Improved control lets programmers elimate redundant contents
- This PDF is little over 300kb including graphics.


# Uses advanced PDF features

ReportLab lets you make use of some of the advanced features of the PDF format that are hard or impossible to use with many other PDF creators.

- Bookmarks and links
- Document Outlines
- Transitions
- PDF Forms
- Embedded fonts


# Now that I've convinced you...

## Hello World

No demo would be right without a simple Hello World style example. I aim to please, so...

*Please see the **hello_world.py** example.*

**Important:** The examples referenced in this document are not good Python code by any stretch of the imagination. They're kept very simple to make what we're demonstrating stand out - at the cost of error handling, good structure, and general quality.

Please don't take them as Python examples, just ReportLab examples.


# The Canvas

ReportLab is based around the idea of a "Canvas", or drawing area. Objects are drawn onto the canvas, either directly or using higher-level drawing APIs.

The canvas follows a co-ordinate scheme more familar to mathematicians than computer programmers.

- Programmers more familar with (0,0) as the top left.
- ReportLab uses the PostScript model more familiar to mathematicians.
- Origin is at the **bottom left** of the page, with positive y moving upwards.
- Objects and text are drawn from their **bottom left**, not top left.


# ReportLab APIs

ReportLab has three major drawing APIs, each with a different purpose and each useful for different things.

- Canvas
- Platypus
- Drawing


## Canvas

- Lowest level API
- Close to the PDF drawing model
- Clumsy for anything complex


# Drawing

- Used for constructing vector graphics
- Like Platypus, uses Canvas for drawing
- Supports non-PDF output formats, too


## Platypus

- Provides a higher level API for document layout
- Includes typesetting features
- Separates formatting and content
- Has useful pre-defined support for tables and other structures
- Uses Canvas for drawing


# A Platypus example

This is straight out of the documentation, and explained better there so I won't cover it in detail.

It demonstrates the use of the automatic text flowing, plus the use of the canvas to draw "extras" like headings and footers.

*Please see the **platypus_demo.py** example.*


# Tables in Platypus

Platypus has very nice support for tables. This is of a great deal of interest when reporting out of a database.

- Easy to construct
- Formatting separated from contents
- Great formatting system well suited to tabular data
- Uses in combination with databases obvious


# Example use of a table

The following program shows how neatly separated formatting and content really are, and how easy it is to build a table.

*Please see the **platypus_table.py** example.*


# PostgreSQL and Python

## Getting data out of the database

The Python DBI is nice to work with

- Simple
- Sensible - tends to do the right thing
- Efficient - lets you get on with programming without worrying about optimisation

I won't go into much detail here, beyond quickly explaining how to get data from the DB.


# PyPgSQL example

As you can see, it's simple to get the data from the DB. The formatting, however, leaves a lot to be desired.

*Please see the **simple_db_example.py** example.*


# Putting ReportLab and the DBI to work

It should be obvious by now that the table formatting abilities of ReportLab's Platypus are a good fit with database interfaces, such as the one provided by PyPgSQL.


# A simple example

The following example demonstrates the simple use of PyPgSQL with Platypus tables. No formatting is attempted.

*Please see the **simple_db_table.py** example.*


# Adding some formatting

It's fairly simple to add formatting to the table without changing the way it's constructed at all.

- Formatting done by a series of rules
- Rules apply to a range of cells
- Later rules override eariler ones, permitting general then specific formatting
- Can control font, type size, font and cell colours, draw lines, etc.

*Please see the **formatted_db_tables.py** example.*


# A real-world example

**customer_bookings_by_month.py** is a real-world example from my work at the POST newspapers. In addition to the table formatting, several other improvements have been made.

- Page number, timestamp and short title in footer
- Uses forms for the static footer info
- Nicely formatted title
- Margins pushed out
- Note the use of the Canvas API for drawing the footer and title
- CGI support


# CGI support

It's simple to create a report that can be used as a normal script or a CGI application.


# Adding CGI support

A few things are needed to make a ReportLab program work as a CGI script.

- Handle getting parameters from a CGI form or command line input
- Detect the CGI environment by looking for **QUERY_STRING** in the environment.


If the CGI environment is found:
Import the cgi and cgitb modules

- print nothing to stdout
- Pass **sys.stdout** as the file argument to the canvas or document constructor.
- Before calling **canvas.save()** or **document.build()**, send some headers.


# Example CGI code

Here are the crucial bits to get CGI support going. First, detect the CGI environment, load CGI support, and create the form object used to read CGI form attributes:

```
    # Detect the CGI environment and load CGI support
    def cgiCheck():
        """Check to see if we're running as a CGI script by checking for th
        QUERY_STRING env var. If found, set ourselves up to run as a CGI pr

        global form,iscgi

        if 'QUERY_STRING' in os.environ:
            iscgi = True
            # Yup, we're running as a CGI program. Load CGI support.
            import cgi
            import cgitb; cgitb.enable()
            form = cgi.FieldStorage()
        else:
            # Nope, looks like we're producing a PDF file instead.
            iscgi = False
```


... set the output file to stdout if we're running under CGI:

```
if iscgi:
    output_file = sys.stdout
else:
    output_file = sys.argv[1]
```

Note that we pass a file object, **sys.stdout**, in the CGI case, and a raw filename otherwise. ReportLab is smart enough to tell the difference and deal with it.

... and send some headers before generating the PDF:

```
# Before sending your output, print some headers
if iscgi:
    print "Content-Type: application/pdf"
    print "Content-Disposition: attachment; filename=report.pdf"
    print
canvas.save()
```


## Converting existing code

To see just how simple it can be to convert an existing app, have a look at **upgrade_odyssey_for_cgi.diff** in the examples. It should apply cleanly to the demos/odyssey/fodessey.py demo in ReportLab 1.11.

I'll leave adding support for getting arguments from an HTML form up to the reader. To run it, you will need the 'odyssey.txt' and possibly full odyssey text in the same directory as the script. Make sure to unzip the full text with the -a argument to unzip!


# Mostly done

That's the basics you need to know covered. A few other issues may be of interest, though, so time permitting we'll get onto those.


# Speed

ReportLab is written almost entirely in Python. As a result you can easily browse over the code, and it's easy to adapt and work with. Unfortunately, it does incur a speed penalty. Speed can be an issue for web use.

## *rl_accel*

A C "accelerator" module in included, but not built by default. Look in **lib/README.extensions** for info about building it. It is used automatically if found.

## *Psyco*

There is also a general purpose JIT (Just In Time) compiler for Python, called Psyco. It's trivial to support in most existing programs.


# Adding Psyco support to an existing app

The follwing code, added after the main import block, will usually result in a significant (often around 30%) speed improvement in Python code.

```
# Try to activate Psyco, failing silently and gracefully if we can't.
try:
    import psyco
    psyco.full()
except:
    pass
```


## A few tips

- **RTFM**. It's an open source app with good documentation. Don't waste it.
- Seriously. Read it cover to cover if you can, it'll save you a lot of wasted time and effort.

## Other uses

- Printing system for legacy apps and languages with limited printing abilities
- Cross-platform printing system for any app - avoids the need to impement a PostScript engine
- Content formatting system for embedding in an app
- Newspaper classifieds pagination. E-Mail me if you're interested in this.


# Extra goodies

## *PythonPoint*

This presentation was written in PythonPoint, a formatting system that converts an XML document to a nicely formatted PDF.

The formatting for this document was shamelessly ripped off from the PythonPoint manual for time reasons. The PythonPoint manual does it much better. Check it out.

## *Py2PDF*

Py2PDF is a cool little app that converts Python source code into syntax-highlighted, neat PDF documents.


# Getting the software

## Python

You'll want at least Python 2.2, with 2.3 recommended. You can get it from http://www.python.org/

## ReportLab

ReportLab is available from http://www.reportlab.org/

## PyPgSQL

The home of PyPgSQL is http://pypgsql.sourceforge.net/


## PIL, the Python Imaging Library

You'll want PIL for working with images in ReportLab. Get it from http://www.pythonware.com/products/pil/ .

## Psyco

Psyco is the Just in Time compiler I (didn't) demonstrate. Get it from http://psyco.sourceforge.net/


# References

- http://www.reportlab.org/os_documentation.html -- The ReportLab docs
- http://www.python.org/peps/pep-0249.html -- The Python DB-API
- The PythonPoint documentation, included in the ReportLab distribution
- http://www.python.org/doc/2.3.3/ -- The excellent Python documentation


# About the Author

Craig Ringer works for the POST Newspapers as a sysadmin, network admin and internal developer. There's a saying about the jack of all trades that may apply to this situation.

He programs in Python by preference, having made a remarkable recovery after learning to program in Perl and Pascal. Any suggestion that Python is "just a scripting language" are best made at more than a 3 metre distance to avoid flying foam.

He can be reached at craig@postnewspapers.com.au, at least until his employer realises he doesn't seem to do anything.