PLATYPUS User Guide and Test Script

Welcome to PLATYPUS!

Platypus stands for "Page Layout and Typography Using Scripts". It is a high level page layout library which lets you programmatically create complex documents with a minimum of effort.

This document is both the user guide & the output of the test script. In other words, a script used platypus to create the document you are now reading, and the fact that you are reading it proves that it works. Or rather, that it worked for this script anyway. It is a first release!

Platypus is built 'on top of' PDFgen, the Python library for creating PDF documents. To learn about PDFgen, read the document testpdfgen.pdf.

What concepts does PLATYPUS deal with?

The central concepts in PLATYPUS are Flowable Objects, Frames, Flow Management, Styles and Style Sheets, Paragraphs and Tables. This is best explained in contrast to PDFgen, the layer underneath PLATYPUS. PDFgen is a graphics library, and has primitive commans to draw lines and strings. There is nothing in it to manage the flow of text down the page. PLATYPUS works at the conceptual level fo a desktop publishing package; you can write programs which deal intelligently with graphic objects and fit them onto the page.

How is this document organized?

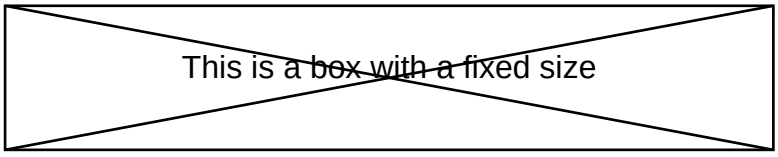
Since this is a test script, we'll just note how it is organized. the top of each page contains commentary. The bottom half contains example drawings and graphic elements to whicht he commentary will relate. Down below, you can see the outline of a text frame, and various bits and pieces within it. We'll explain how they work on the next page.

Now for some demo stuff - we need some on this page, even before we explain the concepts fully

> Platypus is all about fitting objects into frames on the page. You are looking at a fairly simple Platypus paragraph in Debug mode. It has some gridlines drawn around it to show the left and right indents, and the space before and after, all of which are attributes set in the style sheet. To be specific, this paragraph has left and right indents of 18 points, a first line indent of 36 points, and 6 points of space before and after itself. A paragraph object fills the width of the enclosing frame, as you would expect.

Same but with justification 1.5 extra leading and green text.

> Platypus is all about fitting objects into frames on the page. You are looking at a fairly simple Platypus paragraph in Debug mode. It has some gridlines drawn around it to show the left and right indents, and the space before and after, all of which are attributes set in the style sheet. To be specific, this paragraph has left and right indents of 18 points, a first line indent of 36 points, and 6 points of space before and after itself. A paragraph object fills the width of the enclosing frame, as you would expect.



All of this is being drawn within a text frame which was defined on the page. This frame is in 'debug' mode so you can see the border, and also see the margins which it reserves. A frame does not have to have margins, but they have been set to 6 points each to create a little space

Flowable Objects

The first and most fundamental concept is that of a 'Flowable Object'. In PDFgen, you draw stuff by calling methods of the canvas to set up the colors, fonts and line styles, and draw the graphics primitives. If you set the pen color to blue, everything you draw after will be blue until you change it again. And you have to handle all of the X-Y coordinates yourself.

A 'Flowable object' is exactly what it says. It knows how to draw itself on the canvas, and the way it does so is totally independent of what you drew before or after. Furthermore, it draws itself at the location on the page you specify.

The most fundamental Flowable Objects in most documents are likely to be paragraphs, tables, diagrams/charts and images - but there is no restriction. You can write your own easily, and I hope that people will start to contribute them. PINGO users - we provide a "PINGO flowable" object to let you insert platform-independent graphics into the flow of a document.

When you write a flowable object, you inherit from Flowable and must implement two methods. object.wrap(availWidth, availHeight) will be called by other parts of the system, and tells you how much space you have. You should return how much space you are going to use. For a fixed-size object, this is trivial, but it is critical - PLATYPUS needs to figure out if things will fit on the page before drawing them. For other objects such as paragraphs, the height is obviously determined by the available width.

The second method is object.draw(). Here, you do whatever you want. The Flowable base class sets things up so that you have an origin of (0,0) for your drawing, and everything will fit nicely if you got the height and width right. It also saves and restores the graphics state around your calls, so you don;t have to reset all the properties you changed.

Programs which actually draw a Flowable don't call draw() this directly - they call object.drawOn(canvas, x, y). So you can write code in your own coordinate system, and things can be drawn anywhere on the page (possibly even scaled or rotated).

around the contents.

Available Flowable Objects

Platypus comes with a basic set of flowable objects. Here we list their class names and tell you what they do:

XBox This is a contrived object to give an example of a Flowable - just a fixed-size box with an X through it and a centred string.

Paragraph This is the basic unit of a document. Paragraphs can be finely tuned and offer a host of properties through their associated ParagraphStyle.

Preformatted This is used for printing code and other preformatted text. There is no wrapping, and line breaks are taken where they occur. Many paragraph style properties do not apply. You may supply an optional 'dedent' parameter to trim a number of characters off the front of each line.

Image This is a straight wrapper around an external image file. By default the image will be drawn at a scale of one pixel equals one point, and centred in the frame. You may supply an optional width and height.

Table This is a table drawing class; it is intended to be simpler than a full HTML table model yet be able to draw attractive output, and behave intelligently when the numbers of rows and columns vary. Still need to add the cell properties (shading, alignment, font etc.)

Spacer This is a 'null object' which merely takes up space on the page. Use it when you want some extra padding betweene elements.

FrameBreak A FrameBreak causes the document to call its handle_frameEnd method.

Macro This is in progress, but a macro is basically a chunk of Python code to be evaluated when it is drawn. It could do lots of neat things.

Here's the base class for Flowable...

```
s Flowable:
    """Abstract base class for things to be drawn.  Key concepts:
1. It knows its size
2. It draws in its own coordinate system (this requires the
    base API to provide a translate() function.
    """
def __init__(self):
    self.width = 0
    self.height = 0
    self.wrapped = 0

def drawOn(self, canvas, x, y):
    "Tell it to draw itself on the canvas.  Do not override"
    self.canv = canvas
    self.canv.saveState()
    self.canv.translate(x, y)

    self.draw()   #this is the bit you overload

    self.canv.restoreState()
    del self.canv

def wrap(self, availWidth, availHeight):
    """This will be called by the enclosing frame before objects
    are asked their size, drawn or whatever.  It returns the
    size actually used."""
    return (self.width, self.height)
```

The next example uses a custom font

```
        import reportlab.rl_config
            reportlab.rl_config.warnOnMissingFontGlyphs = 0

            from reportlab.pdfbase import pdfmetrics
            fontDir = os.path.join(_RL_DIR,'fonts')
            face = pdfmetrics.EmbeddedType1Face(os.path.join(fontDir,'DarkGardenMK.afm'),
                    os.path.join(fontDir,'DarkGardenMK.pfb'))
            faceName = face.name  # should be 'DarkGardenMK'
            pdfmetrics.registerTypeFace(face)
            font = pdfmetrics.Font(faceName, faceName, 'WinAnsiEncoding')
            pdfmetrics.registerFont(font)


            # put it inside a paragraph.
            story.append(Paragraph(
                """This is an ordinary paragraph, which happens to contain
                text in an embedded font:
                <font name="DarkGardenMK">DarkGardenMK</font>.
                Now for the real challenge...""", styleSheet['Normal']))


            styRobot = ParagraphStyle('Robot', styleSheet['Normal'])
            styRobot.fontSize = 16
            styRobot.leading = 20
            styRobot.fontName = 'DarkGardenMK'

            story.append(Paragraph(
                        "This whole paragraph is 16-point DarkGardenMK.",
                        styRobot))
```

Here are some examples of the remaining objects above.

- This is a bullet point
- Another bullet point

Here is a Table, which takes all kinds of formatting options...

	North	South	East	West
Quarter 1	100	200	300	400
Quarter 2	100	200	300	400
Total	200	400	600	800

We can use images via the file name

```
story.append(platypus.Image('/Users/op/code/reportlab-mirror/tests/pythonpowered.gif'))
story.append(platypus.Image(fileName2FSEnc('/Users/op/code/reportlab-mirror/tests/pythonpowered.
```

They can also be used with a file URI or from an open python file!

```
story.append(platypus.Image('file:///Users/op/code/reportlab-mirror/tests/pythonpowered.gif'))
story.append(platypus.Image(open_for_read('/Users/op/code/reportlab-mirror/tests/pythonpowered.gi
```

This is an ordinary paragraph, which happens to contain text in an embedded font: DarkGardenMK. Now for the real challenge...

THIS WHOLE PARAGRAPH IS POINT DARKGARDENMK.

Images can even be obtained from the internet.

```
img = platypus.Image('http://www.reportlab.com/rsrc/encryption.gif')
story.append(img)
```

Here is an Image flowable obtained from a string filename.

Here is an Image flowable obtained from a utf8 filename.

Here is an Image flowable obtained from a string file url.

Here is an Image flowable obtained from an open file.

We can use images via the file name

```
story.append(platypus.Image('/Users/op/code/reportlab-mirror/tests/gray-alpha.png'))
story.append(platypus.Image(fileName2FSEnc('/Users/op/code/reportlab-mirror/tests/gray-alpha.png
```

They can also be used with a file URI or from an open python file!

```
story.append(platypus.Image('file:///Users/op/code/reportlab-mirror/tests/gray-alpha.png'))
story.append(platypus.Image(open_for_read('/Users/op/code/reportlab-mirror/tests/gray-alpha.png'
```

Here is an Image flowable obtained from a string http url.

JPEGs are a native PDF image format. They should be available even if PIL cannot be used.

Here is an Image flowable obtained from a string filename.

Here is an Image flowable obtained from a utf8 filename.

Here is an Image flowable obtained from a string file url.

Here is an Image flowable obtained from an open file.