PDF-Maker!

Welcome to `pdf-maker.r`'s preliminary documentation. This is my attempt at a PDF generator as a multiplatform solution to printing in **REBOL**. As you can see, it is now quite usable. Current limitations are:

- Sub-optimal image handling (no compression etc.)
- Only the 14 PostScript standard fonts are supported.

Support for custom fonts would definitely require work to be added; justification and right alignment are here thanks to Volker and his Adobe Font Metrics parser.

Following is the documentation for version 1.20 ALPHA

The main function

To create a PDF file (under the form of a REBOL `binary!`), just one function is needed: `layout-pdf`.

```
USAGE:
    LAYOUT-PDF spec

DESCRIPTION:
     Layout a PDF file (based on the provided spec)
     LAYOUT-PDF is a function value.

ARGUMENTS:
     spec -- PDF contents, see documentation for details (Type: block)
```

The spec block is a sequence of blocks, each of them representing a page, so that a document with two pages will look like:

```
layout-pdf [
    [   ; first page
        ; ...
    ] [ ; second page
        ; ...
    ]
]
```

Each page definition may include an optional page size and rotation specification; the page size is expressed in millimeters (as any other position or size in the dialect). For example, to create a page 150 millimeters wide and 140 high, you can write (I'll omit the surrounding `layout-pdf [ ]` from now on):

```
[   page size 150 140
    ; ...
]
```

while to create a page of default size (ISO A4, 211×297 millimeters) but rotated 270 degrees clockwise:

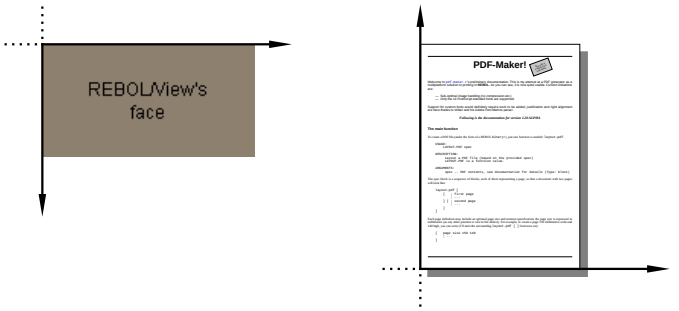
```
[   page rotation 270 ; has to be multiple of 90
    ; ...
]
```

Of course the two can be specified together, in any order:

```
[   page rotation 90 size 180.5 280
    ; ...
]
```

Other than size and rotation, it is possible to specify an offset for the page; this offset is used to translate all the elements inside the page. (This feature has been added to allow fine alignment adjustment when you are printing on paper with a pre-printed form.) To offset a page by 5 millimeters in the *x* axys and 3 millimeters in the *y* axys the command is (as you would have guessed) `offset 5 3`.

Please note that the origin in a PDF page is the **LOWER LEFT** corner and the *y* axys is swapped if you compare it to **REBOL**/View's faces *y* axis:



Page layouts

Each page may contain any number of elements. There are three kinds of elements that can be layed out inside a page: *graphic* elements, *text boxes* and *transformations*. Graphic elements are things such as lines, curves or images; a text box is a rectangular area of the page where it is possible to typeset text. A transformation, instead, is more of an abstract entity; it can be considered as a "context" inside which a geometric transformation is applied to the coordinate system. In other words, you are able to define a new couple of *x* and *y* axys and then render other elements inside this new space.

In this section we will discuss some examples of page layout; a detailed description of all the available commands in the dialect can be found in the following sections.

The simplest useful PDF can be created with the line:

```
layout-pdf [[textbox ["This is some text."]]]
```

(Of course you will probably want to save the result to a file for viewing or printing; `write/binary %file.pdf layout-pdf [...]` will do the job.) The result of the above line is a one page PDF document; this page will just contain the text "This is some text." rendered with the standard font Helvetica with a size of 4.23 millimeters (about 12pt). The default page size (ISO A4) is used, and the text box will have a default size of 191×263 mm and will be positioned at 10 mm from the left edge and 17 mm from the bottom edge of the page. The default paragraph alignment mode is justification; reasonable default values are used for all the parameters.

But now let’s look at another example:

```
layout-pdf [
    [ ; page 1
        line 10 281 201 281
        line 10 16 201 16
    ]
]
```

That will still create a one page document. The page has two black straight lines in it, with a width of 1 mm. The first line goes from (10, 281) mm to (201, 281) mm (remember the origin is at the lower left corner of the page). The second line goes from (10, 16) to (201, 16). These two lines are the ones you can see in the pages of this document.

Let’s try with a layout a bit more complicated.

```
layout-pdf [
    [ ; page 1
        line 10 281 201 281
        line 10 16 201 16
        apply rotation 20 translation 150 255 [
            solid box 200.200.200 edge width 0.2 0 0 26 16
            textbox 3 3 20 10 [
                center font Helvetica 2.8
                "Now with a powerful text typesetter!"
            ]
        ]
    ]
]
```

What does the above do? (You may try looking at the resulting document and then guessing the meaning of the commands used.) The two lines are the same as the previous example; we then used the `apply` command to create a new graphic context. In this context the coordinate axys are rotated by 20 degrees and the origin is translated to the point (150, 255). We are rendering two elements inside this context: a solid box and a text box. The solid box has a color of 200.200.200 (light grey) and an edge with a width of 0.2 mm. (The edge is black, because we did not set a line color before in this page.) The box starts at the axys origin and it’s size is 26×16 mm.

The text box starts at the point (3, 3) and is 20×10. In this text box we are setting the paragraph alignment to `center` and the font to Helvetica 2.8 mm. The paragraph contains just the text “Now with a powerful text typesetter!”. (You have probably already realized that the above is the beginning of this document.)

This should be enough to get you started; but of course the dialect can do more than this. The following sections will address the dialect in detail.

Graphic elements

This section discusses the commands used to render graphics in the page; you can render lines, bézier curves, boxes, circles or any other shape. You can also render sampled images.

We will at first discuss the single commands; at the end we will offer the complete grammar for the graphic elements for reference.

`line`

The `line` command draws a straight line from one point to another, or sets the line options, or both. We already showed how to draw a line using the current options:

`line` *x1 y1 x2 y2*

where *x1*, *y1*, *x2* and *y2* are `number!`s. To set line options, use the command:

`line` *line-options*

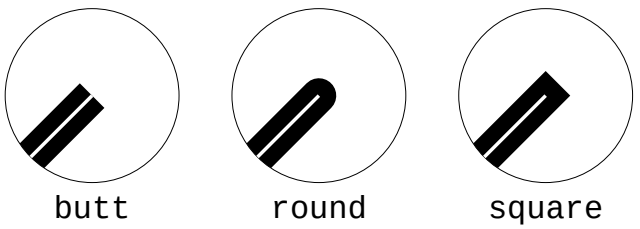
while to do both:

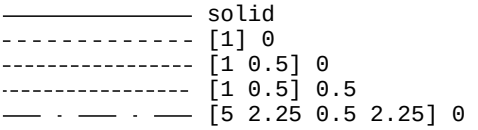
`line` *line-options x1 y1 x2 y2*

where for *line-options* you can specify any of the following:

`width` *line-width*
`cap` *line-cap*
`join` *line-join*
`miter limit` *miter-limit*
`dash` *dash-style*
`color` *line-color*

line-width is the line width in millimeters (0 means the line will be as thin as the output device allows); *line-cap* may be `butt`, `round` or `square`; *line-join* may be `miter`, `round` or `bevel` (this, of course, does not apply to a single line as the ones created by the `line` command; we will discuss this later); *miter-limit* is a `number!` (we'll discuss this later too, together with line join); the *dash-style* may either be the word `solid` (for solid lines) or a `block!` of `number!`s followed by a `number!` (for dashed lines); *line-color* is a `tuple!` expressing the RGB values for the color (the `color` word preceding the tuple is optional). The following figure shows some examples of line cap and dash settings.





As you can see from the example for the line dash settings, the numbers in the block represent, alternatively, the length (in millimeters) of "on" and "off" segments. The block is extended as needed, so that `[1]` means "*1 mm on, 1 mm off, 1 mm on...*" while `[1 0.5]` means "*1 mm on, 0.5 mm off, 1 mm on...*" and so on. The number after the block is used to shift the phase of the dash (i.e. `0.5` means "*shift the dash 0.5 mm to the left*").

`bezier`

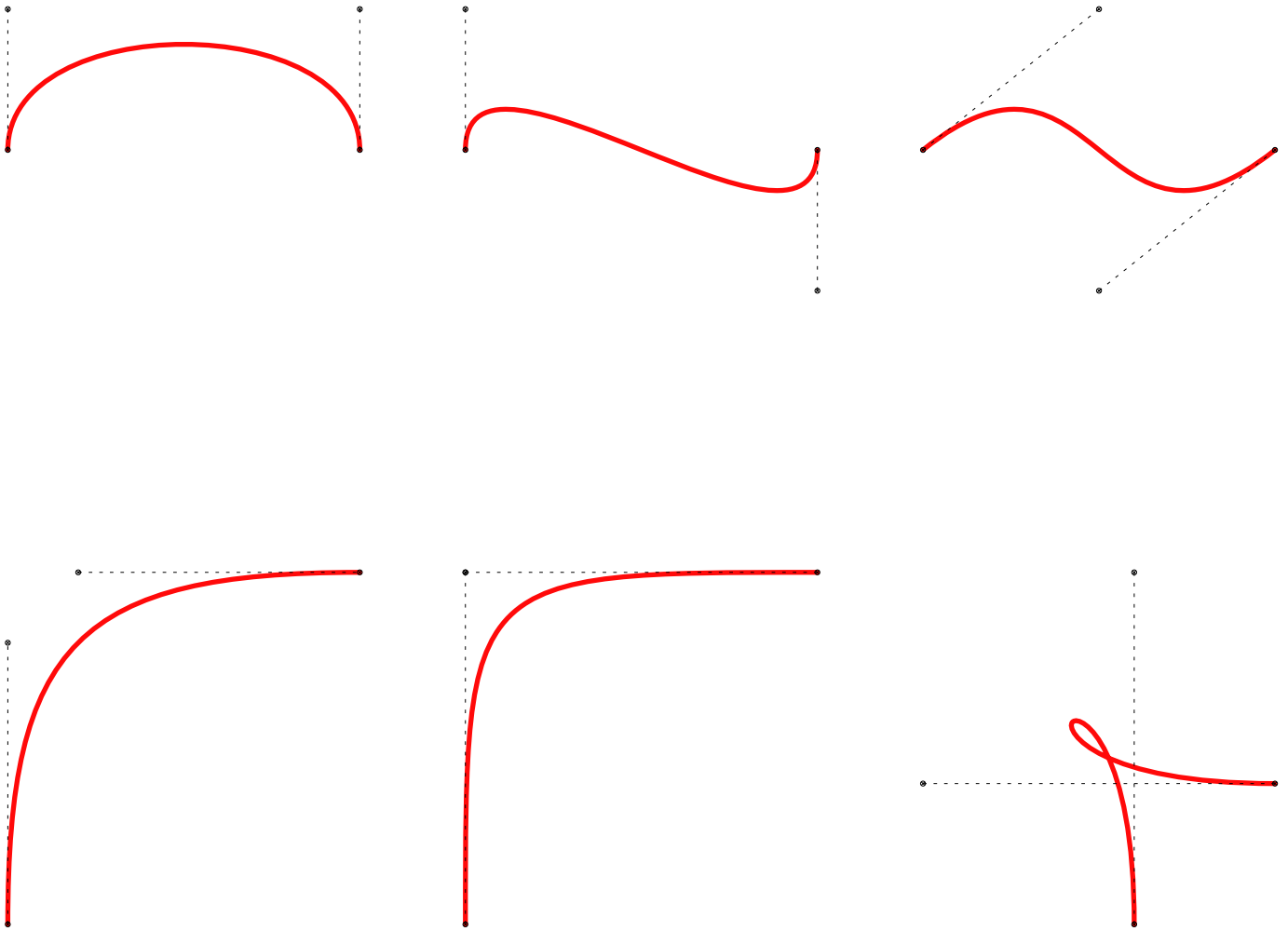
The `bezier` command draws a curve from one point to another. The shape of the curve is controlled by two “control points”. To draw a curve using the current graphic options:

`bezier` *x1 y1 x2 y2 x3 y3 x4 y4*

where (*x1*, *y1*) and (*x4*, *y4*) are the end points, while (*x2*, *y2*) and (*x3*, *y3*) are the control points. Please note that a bézier curve (in general) **does not** pass thru the control points. You can specify line options in the same way you do for the `line` command:

`bezier` *line-options x1 y1 x2 y2 x3 y3 x4 y4*

For more informations on bézier curves you can check a book on computer graphics; here I will only show some examples that can be useful to understand how the control points influence the shape of the curve. The actual curve will be rendered in red, and the points will be evidenced by a circle.



box

The `box` command — as you would expect — draws a rectangle. The syntax is:

`box` *x y width height*

where *x* and *y* are the coordinates of the **LOWER LEFT** point of the rectangle. To give options:

`box` *box-options x y width height*

Box options are the same as line options, except that they are preceded by the word `line` (e.g. you have to write `line width 1` instead of just `width 1`); the only exception is the color option.

solid box

I hate being repetitious, so let's go straight to the point.

`solid box` *x y width height*
`solid box` *solid-box-options x y width height*

The difference between the options for a `solid box` vs. the ones for a `box` is that the options for the edge of the box are preceded by the `edge` word (instead of `line`). You can specify two colors: one for the edge (preceded by the `edge` word) and one for the body of the rectangle. Maybe it's better with some examples:

```
edge width 0.5 edge
  0.0.0 250.240.0
```

```
edge width 0 edge
255.0.0 0.255.255
```

circle and solid circle

These commands approximate a circle with béziers. The approximation is probably not very good, but should be acceptable for most purposes. The syntax is:

`circle` *circle-options x y radius*
`solid circle` *solid-circle-options x y radius*

where *x* and *y* are the coordinates of the center; with regard to the options, they are exactly the same as for `box` and `solid box`.