Ruby Ghostscript

Document Builder

v0.8

Ruby Ghostscript - Document Builder
http://rubyforge.org/projects/rghost/

Author: Shairon Toledo
shairon.toledo@gmail.com
http://www.hashcode.eti.br
Brazil Jun/2008

Summary

1. About

Ruby Ghostscript (RGhost) is a library for document developers wanting a quick and easy way to generate pdf files. Notable features include: inserting images, vector drawing , text,font support , EPS template support and multiple output formats. RGhost acts as a Ruby wrapper over the Ghostscript engine enriched by a predefined set of Postscript(ps) functions. For example:

In Ruby code you'll write:

```
doc.horizontal_line :middle, :start_in => 3, :size => 2
```

This gets translated into the following Postscript function. Note how this is not actually raw Postscript; instead, it makes use of a set of predefined functions shipped with RGhost.

```
3 unit 2 unit exch gsave current_row row_height sqrt add moveto 0 rlineto stroke grestore
```

Unless embarking into deep Postscript wizardry you don't need to be aware of this fact.

Basically RGhost is a helper for creating Postscript documents. The resulting Postscript document is rendered to the target format (usually pdf but not just) by invoking Ghostscript, the free Postscript interpreter. Ghostscript settings vary a lot depending on your platform. Initially RGhost was developed for *nix environments where Ghostscript is (almost always) native and used as a printing filter (CUPS, Windows Print Service, LPRng etc) and also as a document format converter.

During the conversion of the postscript code to the desired format, four files are created, three of them being temporary.

- **input:** The input file is a pure postscript file with a **.rgin** extension that will be automatically removed after the conversion process, even if errors show up.
- **errors:** Using a **.rgerr** extension, its content are the errors generated by ghostscript. It'll be deleted after the conversion. The content of the file is available on the **errors** variable of the **RGhost::Engine** class.
- **log:** The log file is set using the option **:logfile** of the RGhost::Engine class, appending all logs to it. Disabled by default.
- **output:** The output files will have the extension passed to the **RGhost::Engine#render** method. For multi-page formats it will return an array of files. These files **aren't** deleted automatically, meaning that the developer will have to deal with them.

2. Installation

Install the GPL Ghostscript PostScript/PDF interpreter for your platform, which can be GPL or ESP. It available at this site: Ghostscript, Ghostview and Gsview. After Ghostscript installation, install rghost gem; if you need barcodes install the rghost_barcode optional package.

```
shell# gem install rghost
```

For rghost_barcode:

```
shell# gem install rghost_barcode
```

3. Configuring

By default, the variable RGhost::Config::GS[:path] has the content '/usr/bin/gs', but you must configure it if your gs binary is in another path. For example, in the Microsoft platform.

```
RGhost::Config::GS[:path]= 'C:\\gs\\bin\\gswin32c.exe'
```

Listed below are the keys of the said hash.

Key	Example	Description
:path	`RGhost::Config::GS[:path]='C:\\gs\\bin\\gswin32c.exe'`	Path to executable ghostscript.
:tmpdir	`RGhost::Config::GS[:tmpdir] = ENV['TEMP']`	Temporary directory

:default_params	`RGhost::Config::GS[:default_params] << '-dSAFER'`	Allows you to add/remove options. (use with caution!)
:stack_elements	`RGhost::Config::GS[:stack_elements]=5000`	Defines the maximum number of elements for each matrix inside postscript's internal stack, avoiding a stack overflow error.
:unit	`RGhost::Config::GS[:unit]=RGhost::Units::Cm`	Set the measure units. See Units::Unit for available units.
:charset_convert	`RGhost::Config::GS[:charset_convert]= lambda {|text| Iconv::iconv('latin1','utf8', text)}`	Ruby to PS character conversion proxy. Necessary when the source code isn't in the same encoding as the document. *Params* is a block that returns a String. The default setting is **nil**.
:font_encoding	`RGhost::Config::GS[:font_encoding]= :IsoLatin`	Sets the Postscript font encoding. Default: **:IsoLatin.** This parameter can be enabled at the creation of the document.
:extensions	`RGhost::Config::GS[:extensions] << “/mydir”`	Sets the options library path. Includes fonts and postscript codes.
:preload	`RGhost::Config::GS[:preload] << :mylib`	Preloads library by file name, the system will search inside **:extensions** directories.

Normally only the **:path** is necessary.

3.1 Environment test

Your first code to test your environment.

```
require 'rubygems'
require 'rghost'
RGhost::Config.is_ok?.render :pdf, :filename => "/tmp/mytest.pdf"
```

One the page, it looks like this

This will be shown if everything is right :).
In Windows

```
require 'rubygems'
require 'rghost'
RGhost::Config::GS[:path]= 'C:\\gs\\bin\\gswin32c.exe'
RGhost::Config.is_ok?.render :pdf, :filename => "/tmp/mytest.pdf"
```

4. Basic usage

All objects inherit the PsObject class, so they all share the main methods **set, raw, call** and **ps**. To create a PSObject you pass a string (or block). Example:

Using postscript commands

```
ps=PsObject.new(' 1 5 2 mul add')
```

Calling an object, function or variable directly from the postscript stack

```
ps=PsObject.new
ps.call :showpage
```

Writing directly to the stack

```
ps=PsObject.new
ps.raw '(string text) show'
```

Creating a PSObject in a block.

```
ps=PsObject.new do
  raw '1 5'
  raw '2'
  raw 'mul'
  raw 'add'
end
```

The main class is Document. The Document class is a PsFacade; most methods are defined in this class. Normally in examples you will see the doc.*method options ... .* You can work with RGhost in two ways.
(recommended)

```
require 'rubygems'
require 'rghost'
include RGhost
doc=Document.new
```

or

```
Require 'rubygems'
require 'rghost'
doc=RGhost::Document.new
```

4.1 Units

The postscript default unit is 1 pixel/72 inch. RGhost uses centimeters (**cm**) as its default, mainly for positioning coordinates and numeric object sizes (if the size is a String, it won't be parsed to the default unit). This setting can be changed setting the value of RGhost::Config::GS[:unit] before the document is created using any of the Units child classes. Example:

Setting to inches.

```
RGhost::Config::GS[:unit]=Units::Inch
doc.moveto :x => 1, :y => 2  #1 inch  x e 2 inches  y
```

Explicitly setting to Cm.

```
doc.moveto :x => '1 cm' , :y => '2 cm'
```

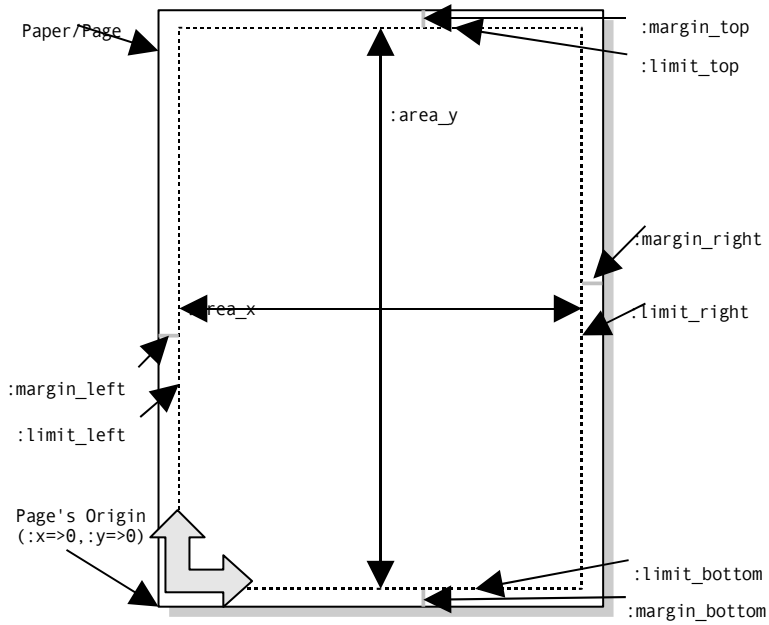
or using the Postscript unit

```
doc.moveto :x => '100' , :y => '200'
```

4.2. Paper and Page Layout

4.2.1 Paper

Paper is the area where the Postscript elements/objects are drawn. The margin is the document's non-printable area (both by the cursors and by the page's internal controllers). The :area_x and :area_y are the printable section of the paper.



The nouns come before the adjectives as in CSS, like this: margin-top, margin-top, etc.
You can specify an equal margin on all sides, like this:

```
doc=Document.new :paper :A5, :margin => 1
```

Defining custom margins

```
 doc=Document.new :paper => :A5, :margin => [0.5, 1, 0.5, 2] #=>[top,right,bottom,left]
```

Or specify individually as in the example below:

```
doc=Document.new :paper => :A4, :margin_left => 3
```

To use a landscape orientation just pass **true** to the :landscape **option**. Example:

```
doc=Document.new :paper => :A4, :landscape => true
```

The same goes for duplex printing:

```
doc=Document.new :paper => :letter, :duplex => true, :tumble => false
```

More info on preset paper sizes on [Paper sizes known to Ghostscript](http://www.cs.wisc.edu/~ghost/doc/cvs/Use.htm#Known_paper_sizes)
http://www.cs.wisc.edu/~ghost/doc/cvs/Use.htm#Known_paper_sizes

4.2.2 Custom Paper

Just pass the size as an array (width x height) and it will be sent to GS.

```
doc=Document.new :paper => [15,10]
```

4.3 Object orientation and position

There are two ways that you can work; the relative position with automatic cursor and absolute position passing the coordinate x,y to object or specifying in the body of the document.

4.3.1 Relative

4.3.1.1 next_row()

Jump to the next row. It's the same as jump_row(1).

```
doc=Document.new
doc.show "Row 1"
doc.next_row
doc.show "Row 2"
doc.next_row
doc.show "Row 3"
```

4.3.1.2 show_next_row(text,show_options)

Executes **show** and the method **next_row**. The result of the code below is the same as above.

```
doc=Document.new
doc.show "Row 1"
doc.show_next_row “Row 2”
doc.show_next_row “Row 3”
```

4.3.1.3 goto_row(row_number)

The class method **goto** positions the cursor based on page row number. Example:

```
d=Document.new
d.goto_row 15
d.show "You're on row 15"
d.goto_row 3
d.show "Now you're on row 3"
```

Or without facade

```
d=Document.new
d.set Cursor.goto_row(15)
d.set Show.new(" You're on row 15")
d.set Cursor.goto_row(3)
d.set Show.new("Now you're on row 3")
```

4.3.1.4 jump_rows(rows)

Jumps n rows relative to the current row

```
d=Document.new
d.jump_row 4    # jump four rows below
d.jump_row -5   # go back up five rows
```

4.3.2 Absolute

4.3.2.1 moveto(point={:x => :limit_left, :y => :current_row)

Moves cursor to absolute point relative to default source point x=0 and y=0 of the page. It doesn't interfere with the row positions.

```
doc=Document.new
doc.moveto :x=> 10, :y=> 5
doc.show "Hello Girls!!!"
```

4.3.2.2 rmoveto(point={:x => :limit_left, :y => :current_row)

It works the same way as **moveto**; the only difference is that it's relative to the current point.

```
doc=Document.new
doc.moveto :x=> 10, :y=> 5
doc.show "Hello Girls!!!"
doc.rmoveto :x => 5               # move to x=> 15 (10 plus 5) maintaining y => 5
```

4.3.2.3 translate(point={:x =>0 , :y => 0})

It changes the default point to a new point(dislocate)

```
doc=Document.new
doc.translate :x=> 2, :y=> 1
#the translate method dislocates to :x=> 2, :y=> 1, so the moveto(0,0) will refer to the
same    point (2,1).
doc.moveto :x => 0,:y => 0
doc.translate :x=> -2, :y=> -1  # returns to default source point
```

4.3.2.4 rotate(angle)

Rotates all objects after executing them, passing the angle as argument.

```
d=Document.new
d.rotate 90
#do something
d.rotate -90  # goes back to source angle
```

Using it inside **newpath** block

```
d=Document.new
d.newpath do
  rotate 90
  show 'Foo'
end
```

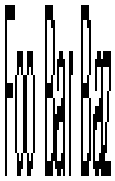
Using it inside **graphic** block

```
d=Document.new
d.graphic do
  rotate 90
  show 'Foo'
end
```

4.3.2.5 Zoom and Scale

```
doc.scale(1,1)      #default document scale
```

```
doc.scale(1,3)
```



```
doc.scale(3,3)
```

Foo Bar Baz

Scales proportionally by value in percent. Example

```
doc.zoom(300)   # 300% the same as scale(3,3)
```

4.3.3 Page control

4.3.3.1 next_page

It goes to the next page and resets the cursor values(recommended).

```
doc=Document.new
doc.show "Page 1 row 1"
doc.next_page
doc.show "Page 2 row 1"
doc.next_page
doc.show "Page 3 row 1"
```

4.3.3.1 showpage

It goes to the next page without resetting the cursor values. Often used for single page documents.

```
doc=Document.new
doc.show "Page 1 row 1"
doc.showpage # page 2, but internally page 1
doc.show "Page 1 row 1"
doc.showpage # page 3
doc.show "Page 1 row 1"
```

5. Fonts

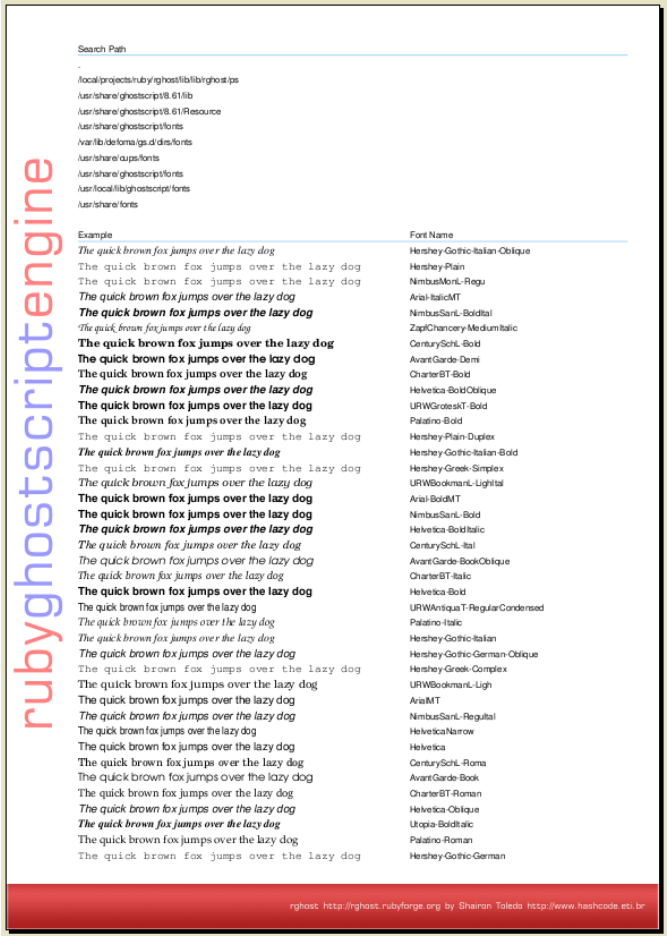
The list of available fonts depends on your platform; by default the following are available:

AvantGarde-Book	Helvetica-Narrow-BoldOblique
AvantGarde-BookOblique	Helvetica-Narrow-Oblique
AvantGarde-Demi	Helvetica-Oblique
AvantGarde-DemiOblique	NewCenturySchlbk-Bold
Bookman-Demi	NewCenturySchlbk-BoldItalic
Bookman-DemiItalic	NewCenturySchlbk-Italic
Bookman-Light	NewCenturySchlbk-Roman
Bookman-LightItalic	Palatino-Bold
Courier	Palatino-BoldItalic
Courier-Bold	Palatino-Italic
Courier-BoldOblique	Palatino-Roman
Courier-Oblique	Times-Bold
Helvetica	Times-BoldItalic
Helvetica-Bold	Times-Italic
Helvetica-BoldOblique	Times-Roman
Helvetica-Narrow	ZapfChancery-MediumItalic
Helvetica-Narrow-Bold	

To discover all fonts in your environment generate the catalog.

```
RGhost::Config.environment_fonts.render :pdf, :filename => 'mycatalog.pdf'
```

it generates the page



Fonts are addressed by their names so you must use the font name to define your tags.

5.1 Using external fonts

To use external fonts, the font must be in TTF, PFB or PFA font format. Below are the step-by-step instructions.
First, create the directory, in this case *foo*

```
shell# mkdir /tmp/foo
```

Copy the fonts to /tmp/foo dir. Inside *foo* dir create the file Fontmap(it must be this name); in body of file make the following font map to Trebuchet.ttf and Verdana.ttf.

```
/Trebuchet (Trebuchet.ttf);
/Verdana   (Verdana.ttf);
```

thus

```
shairon@hashcode:/tmp/foo$ ls -lh
-rw-r--r-- 1 shairon shairon   57 2008-06-12 17:25 Fontmap
-rw-r--r-- 1 shairon shairon 124K 2008-06-12 17:23 Trebuchet.ttf
-rw-r--r-- 1 shairon shairon 137K 2008-06-12 17:24 Verdana.ttf
```

Loading the fonts

```
RGhost::Config::GS[:extensions] << '/tmp/foo'
```

Now you have to define tags using the method **define_tags**

```
d.define_tags do
 tag :myfont, :name => 'Verdana', :size => 12, :color => '#ADAD66'
end
```

Using the tag

```
 doc.show 'Hello using the Verdana font', :tag => :myfont
```

with method/class Text

```
doc.text 'Hello using the <myfont>Verdana</myfont>'
```

5.2 Defining tags

The Document#define_tags method creates a map of tags which will be used in 'writable' classes (Show, Text, TextIn and TextArea). The font names file catalog can be generated by the code below.

```
 RGhost::Config.environment_fonts.render :pdf, :filename => 'mycatalog.pdf'
```

This can take a while. If it takes too long your are probably having font problems. Remove some fonts, particularly little used international fonts. Below is a little piece of the catalog:

Example	Font Name
The quick brown fox jumps over the lazy dog	Hershey-Gothic-Italian-Oblique
The quick brown fox jumps over the lazy dog	Hershey-Plain
The quick brown fox jumps over the lazy dog	NimbusMonL-Regu
The quick brown fox jumps over the lazy dog	Arial-ItalicMT
The quick brown fox jumps over the lazy dog	NimbusSanL-BoldItal

After generating your catalog you can map your tags.

Tags have the name of tag(as Symbol) and its options. The options are

:name - Font name from catalog.

:size - Font size.

:color – Color.create facade

:encoding - If true the font will be encoded using the pattern :font_encoding of the document.

Examples

```
d=Document.new :encoding => 'IsoLatin'
d.define_tags do
 tag :my_italic,    :name => 'Hershey-Gothic-Italian-Oblique', :size => 10
 tag :myfont,       :name => 'Hershey-Plain'
 tag :font_encoded, :name => 'NimbusMonL-Regu', :size => 8, :color => 0.5,:encoding => true
 tag :other_font,   :name => 'NimbusMonL-Regu',    :size => 10
 tag :arial,        :name => 'Arial-ItalicMT',     :color => '#ADAD66'
 tag :arial_bold,   :name => 'NimbusSanL-BoldItal',:size => 12, :color => '#ADAD66'
end
```

You can use the **:default_font** tag for customizing the default font.

5.3 Using tags

With **Show** class

```
doc.show 'My Text on this row', :with => :my_italic, :align => :page_center
```

With **Show** class overriding the tag‘s color.

```
doc.show 'My Text on this row', :with => :my_italic, :align => :page_center, :color => :red
```

With **TextIn** class.

```
doc.text_in :x=> 3, :y=> 10, :tag => :arial_bold , :write => "Here's point(3,10)"
```

With **Text**

```
 doc.text '<myfont>My Text</myfont>on this row.<arial>Other text</arial><my_italic>Italic
font</my_italic>'
```

With **TextArea**

```
 txt='<myfont>My Text</myfont>on this row.<arial>Other text</arial><my_italic>Italic
font</my_italic>'
 doc.text_area txt, :text_align => :center, :width => 5, :x => 3, :y => 10
```

5.4 use_tag method

```
doc.use_tag :myfont
doc.show "Simple Text", :tag => nil # it will use :myfont
doc.show "Simple Text2"   # it will use :myfont too
```

5.5 Default tags

The default tags are defined in the constant RGhost::Config::FONTMAP, below the content of the block.

```
RGhost::FontMap.new :name => "Helvetica", :size => 8, :encoding => false do
    new :span
    new :b,     :name => "Helvetica-Bold"
    new :bold,     :name => "Helvetica-Bold"
    new :normal,   :name => "Helvetica"
    new :i,     :name => "Helvetica-Oblique", :size => 8
    new :bi,    :name => "Helvetica-BoldOblique"
    new :big,   :size => 10
    new :small, :size => 7
    new :h1,    :name => "Helvetica", :size => 14
    new :h2,    :name => "Helvetica", :size => 13
    new :h3,    :name => "Helvetica", :size => 12
    new :h4,    :name => "Helvetica", :size => 11
    new :h5,    :name => "Helvetica", :size => 10
    new :title, :name => "Helvetica", :size => 20
    new :pre,   :name => "Courier"
  end
```

You can customize the tag by overriding tags with the same name. Example for tag **b.**

```
d=Document.new :encoding => 'IsoLatin'
d.define_tags do
  new :b,  :name => "Helvetica-Bold", :font => 12
end
```

The tag **:normal** is default if is not specified.

6. Printing text

6.1 Show

Writes a text on the current row or points with align. Options:

:tag or **:with** - Uses predefined tag.
:color - Overrides color of the tag.
:align - Text align.

Examples

The vertical red line is the current point. To align by point

```
doc.moveto :x => 3, :y => 4
doc.show "Foo Bar Baz", :align => :show_left      #default
```


```
doc.moveto :x => 3, :y => 4
doc.show "Foo Bar Baz", :align => :show_center
```


```
doc.moveto :x => 3, :y => 4
doc.show "Foo Bar Baz", :align => :show_right
```

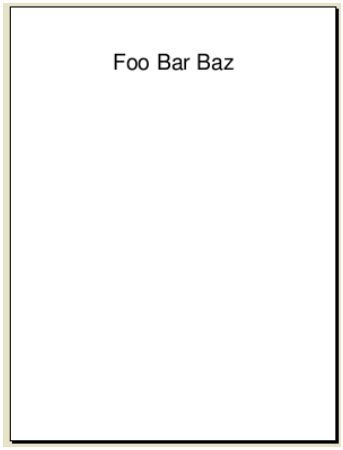
For the current row it's not necessary to position using moveto. As below:

```
doc.show "Foo Bar Baz", :align => :show_right
```

Now page justification

```
doc.show "Foo Bar Baz", :align => :page_left
```

```
doc.show "Foo Bar Baz", :align => :page_center
```




```
doc.show "Foo Bar Baz", :align => :page_right
```

Overriding tag color:

```
doc.show "Foo Bar Baz", :with => :my_italic, :align => :page_center, :color => :red
```

Many tags per row. Defining tags

```
doc=Document.new
doc.define_tags do
  tag :font1, :name => 'Helvetica', :size => 10, :color => '#F34811'
  tag :font2, :name => 'Times',     :size => 14, :color => '#A4297A'
  tag :font3, :name => 'TimesBold', :size => 18, :color => '#AA3903'
end
```

using it

```
doc.show "foo bar baz ",  :with => :font1
doc.show "qux quux ",     :with => :font2
doc.show "corge ",        :with => :font3
doc.show "grault garply ",:with => :font2
doc.show "qux quux",      :with => :font1
```

6.2 TextIn

TextIn is a helper to combine cursor positioning and text output into one step. Options:

- **`:x`** and **`:y`** - Initial position.
- **`:tag`** or **`:with`** - Use predefined tag.
- **`:color`** - Override color of the tag.
- **`:text`** or **`:write`** - The text.

Examples

```
doc=Document.new
doc.text_in :x => 3, :y => 4, :write => "Foo Bar Baz", :tag => :h1
```

Rotating

```
doc.newpath do
 translate :x => 3, :y=> 4
  rotate 45
  text_in :x => 0, :y => 0, :write => "Foo Bar Baz1", :tag => :font2
end
```

Eval postscript internal

TextIn will evaluate postscript internal variables you pass in between % signs. Sounds complex, huh? Let's see an example:

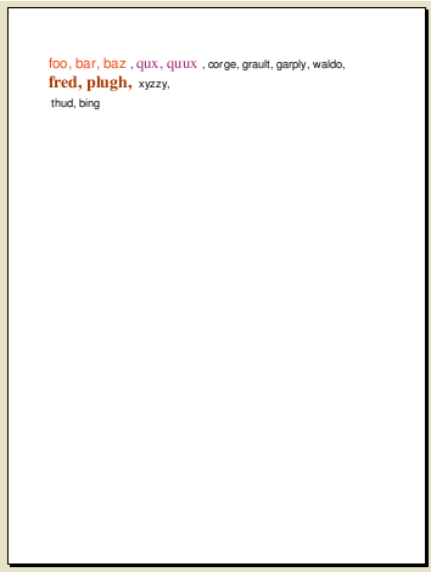
```
doc.text_in :x=> 3, :y=> 5.5, :text => "this is %row% row and current page %current_page%"
```

6.3 Text

Wraps the text so as the it fits on the page(:area_x). Wrapping happens in whitespace characters without hyphenation. Additionally you can make use of predefined tag and the special tag
 to break row. You can disable the parse with second parameter tag_parse=false.

Examples

```
doc=Document.new
doc.define_tags do
  tag :font1, :name => 'Helvetica', :size => 10, :color => '#F34811'
  tag :font2, :name => 'Times',     :size => 11, :color => '#A4297A'
  tag :font3, :name => 'TimesBold', :size => 12, :color => '#AA3903'
end
my_text="<font1>foo, bar, baz</font1>,<font2>qux, quux</font2>, corge, grault, garply,waldo,
<font3>fred, plugh,</font3> xyzzy,<br/> thud, bing"
doc.text my_text
```



Without parse

```
 atext="<font1>foo, bar, baz</font1>,<font2>qux, quux</font2>, corge, grault, garply, waldo,
<font3>fred, plugh,</font3> xyzzy,<br/> thud, bing"
  doc.text atext,false
```

6.4 TextArea

TextArea wraps the text so that it fits in a box of a given width. Wrapping happens in whitespace characters without hyphenation. Additionally you can make use of predefined tag and the special tag
 to break row. The alignment can be left, right and centered.

PS: It doesn't jump pages.

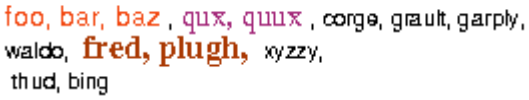
Options

- **`:x`** and **`:y`** - Initial position.
- **`:row_height`** - Row height :)
- **`:width`** - Maximum width of the text
- **`:text_align`** - Align the text in the virtual box using :left, :right and :center.

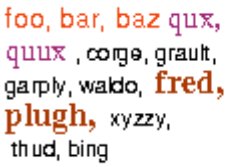
Examples

```
doc=Document.new
my_text="<font1>foo, bar, baz</font1><font2>qux, quux</font2>, corge, grault, garply,
```

```
waldo, <font3>fred, plugh,</font3> xyzzy,<br/> thud, bing"
 doc.text_area my_text
```

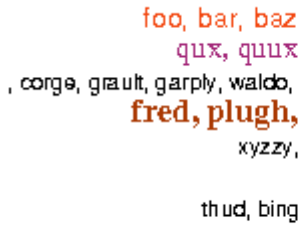



```
doc.text_area my_text, :width =>3
```

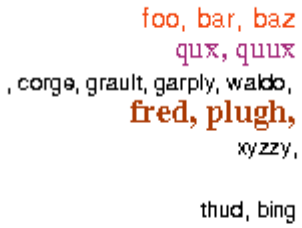



```
doc.text_area my_text, :width =>3, :text_align => :center
```


```
doc.text_area my_text, :width =>3, :text_align => :right
```




```
doc.text_area my_text, :width =>3, :text_align => :right, :x => 3
```




```
doc.text_area my_text, :width =>3, :text_align => :right, :x => 3, :row_height => 0.6
```

6.6 Printing text from file

Prints the text file using the predefined tag **:pre**

Example

```
doc=Document.new :paper => :A4, :landscape => true
doc.print_file "/etc/passwd"
doc.render :pdf, :filename => "/tmp/passwd.pdf
```

or

```
doc=Document.new :paper => :A4, :landscape => true
File.readlines("/etc/passwd").each {|line| doc.show_next_row(line) }
doc.render :pdf, :filename => "/tmp/passwd.pdf
```

Another way

```
doc=Document.new :paper => :A4, :landscape => true
doc.text File.readlines.join('<br/>')
doc.render :pdf, :filename => "/tmp/passwd.pdf
```

7. Working with templates and images

7.1 Template

RGhost can make use of Encapsulated Postscript files to act as templates (EPS). This way you can create the visual layout of the page using a graphics tool and just paint the dynamic pieces over using RGhost.

mytemplate.eps

my_ruby_program.rb

output [ps, pdf, jpeg, png, tif, pcl, deskjet, laserjet, etc]

Above we have mytemplate.eps that was generated by a graphic app, my_ruby_program.rb that takes care of the positioning and ultimately the generated output.

A Template use example: Let's say that the files first.eps and content.eps already exist. Now we will see how to create a document that uses the template first.eps for the cover and the rest of the document uses content.eps.

```
d = Document.new :margin_top => 5, :margin_bottom => 2
```

Just for the first page

```
d.first_page do
 image "/my/dir/first.eps"                          #loads the template
 text_in :x=> 5, :y=> 17, :text => "My Report", :with => :big
 next_page                                          #go to the next page using cursors
end
```

Callback for all other pages.

```
d.before_page_create :except => 1 do
 image "/my/dir/content.eps"
 text_in :text => "Page %current_page% of %count_pages%", :x => 18, :y => 27, :with =>
```

```
:normal
 end
```

1500 rows

```
 1500.times do |n|
   d.show_next_row "Value #{n}"
 end
```

We have a cover page and 1500 rows, judging by the margins each page supports 46 rows, so we have 1500/46 = 32.60 pages plus the cover. Rounding it up totals 34 pages for the :count_pages

```
  d.define_variable(:count_pages, 34)
  d.showpage
  d.render :pdf, :filename => "/tmp/test.pdf"
```

If we knew the number of pages beforehand we could state it on the creation of the document, i.e.

```
  :count_pages => 34
```

The example uses one template per page, but this is not a limit in RGhost. You can have multiple images and templates per page. Just have to define the template:

```
   d=Document.new :margin_top => 5, :margin_bottom => 2
   d.define_template(:myform, '/local/template/form1.eps', :x=> 3, :y => 5)
```

and call it on the document.

```
  d.use_template :myform
```

Arguments

:name - Template name.
:file_path - Path to file.
:options - Options facade to Image.for(or image)

7.2 Image.for

Facade method for load image by file extension. Uses Eps, Gif and Jpeg class. Accepts gif, jpeg, jpg and eps extensions.

Options:

:x and **:y** - Coordinates to position.
:rotate - Angle to image rotation if there is one.
:zoom – Resize the image proportionally.

7.2.1 EPS

Loading eps file

Examples

```
doc=Document.new
doc.set Eps.new "/local/templates/myform.eps", :x => 10, :y => 3
```

Using Image.for facade

```
doc.set Image.for "/local/templates/myform.eps", :x => 10, :y => 3
```

Using PsFacade or Document.

```
doc.image "/local/templates/myform.eps", :x => 10, :y => 3
```

7.2.2 Gif

Loads GIF image from file

Examples

```
doc=Document.new
```

```
doc.set Gif.new "../public/images/button.gif", :x => 10, :y => 3
```

Using Image.for facade

```
doc.set Image.for "../public/images/button.gif", :x => 10, :y => 3
```

Using PsFacade or Document.

```
doc.image "images/button.gif", :x => 10, :y => 3
```

Using 200 percent zoom

```
doc.image "images/button.gif", :zoom => 200
```

7.2.3 Jpeg

Loads JPEG image from file

Examples

```
doc=Document.new
doc.set Jpeg.new "../public/images/button.jpg", :x => 10, :y => 3
```

Using Image.for facade

```
doc.set Image.for "../public/images/button.jpg", :x => 10, :y => 3
```

Using PsFacade or Document.

```
doc.image "images/button.jpg", :x => 10, :y => 3
```

Using Zoom of the 200 percent

```
doc.image "images/button.jpg", :zoom => 200
```

8. Document and Callbacks

Callbacks are custom code blocks defined inside Document. All callbacks implicitly receive an instance of PsFacade to create any PsObject. The callbacks‘ execution depends on the algorithms predefined in Postscript core library. There are two kinds of callbacks, Static and Dynamic callbacks.

In a Static callback there aren‘t parameters to control the inclusion and exception in the current scope, which is usually applied to events which happen one time, for example after_document_create: "after document create" will always execute once for each document. Otherwise, with Dynamic callbacks there is control of scope using conditional :only or :except; this is the only difference in relation to static callbacks. The parameters of a Dynamic callback must be one integer or one array of integers.

Example: For all pages except page 3

```
doc.before_page_create :except => 3 do
    # do something
end
```

For just 2 and 4 pages

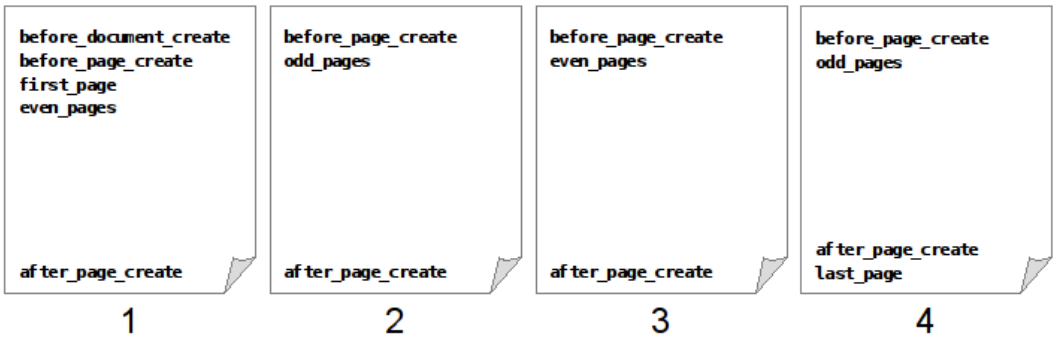
```
doc.before_page_create :only => [2,4] do
    # do something
end
```

Most callbacks are defined in facades such as DocumentCallbackFacade and Grid::CallbackFacade
This module is included inside Document class. It will create methods to make it easier to use.

8.1 Document callback execution order

Example for a document with 4 pages; looking at the picture you can see the order in which the callbacks are

executed.



8.2 Page numbering

You can combine the callback before_page_create and text_in for product page numbering using internal variables.
Example:

```
doc=Document.new :count_pages => 3
doc.before_page_create :except => 1 do
  text_in :x=> 3, :y=> 5.5, :text => "%current_page% of %count_pages%"
end

doc.next_page
doc.next_page
doc.next_page
```

8.3 Benchmarking

Starts and Ends internal benchmark will write at bottom of page.

Example

```
doc=Document.new
doc.benchmark :start
doc.... #do something
doc.benchmarck :stop
doc.render ...
```

8.4 render

Facade to RGhost::Engine#render converts a document to an output format, such as :pdf, :png, :ps, :jpeg, :tiff etc The parameter device can be found at RGhost::Constants::Devices or at pages.cs.wisc.edu/~ghost/doc/cvs/Devices.htm

Options
Method render has the following options available.

:filename - File path.
:logfile - Writes the converter's process into a file.
:multipage - if true the output will be one page per file posfixed by _0001.ext, for example, one file name 'test.png' with two pages will create test_001.png and test_002.png
:resolution - Integer value for output resolution.
:size - Crops a single page using a string of dimensions, example, '200x180', '140x90'.
:range - Specifies range of pages (PDF only)

8.4.1 Ghostscript interpreter options

Arrays of Hashes for Ghostscript interpreter. Look at pages.cs.wisc.edu/~ghost/doc/cvs/Use.htm#Parameter_switches for more details. You can use two parameter **:s** and **:d**, examples

```
   :s => [{:GenericResourceDir => /dir, :DEFAULTPAPERSIZE=> "a3"}]
   :d => [ {:TextAlphaBits => 2}  ]
```

Or one string using the parameter :raw, as below

```
   :raw => "-sGenericResourceDir=/test -dTextAlphaBits=2"
```

Examples

```
doc=Document.new
  #do something
doc.render :pdf,  :filename => 'foo.pdf   # PDF output
doc.render :jpeg, :filename => 'foo.jpg'  # JPEG output
doc.render :png,  :filename => 'foo.png',  :multipage => true # PNG output one page per file
doc.render :tiff, :filename => 'foo.tiff', :resolution => 300      # TIFF with 300dpi
doc.render :ps, :raw => '-sFONTMAP=/var/myoptional/font/map', :filename => 'test.ps'
```

8.4.2 Testing if it has errors

```
doc=Document.new
doc.raw "hahahah!" #it produced an error in ps stack
doc.render :jpeg, :filename => 'with_error.jpg'
puts r.errors  if r.error?
#=> GPL Ghostscript 8.61: Unrecoverable error, exit code 1.\ Error: /undefined in hahahah!
```

8.4.3 Printing

Using printing system

```
  doc=Document.new
    #do something
  f="myjob.prn"
  doc.render :laserjet, :filename => f
  `lpr #{f}`
```

8.4.4 Windows shared printer

```
doc.render :eps9mid, :filename => "//machine/printer"
```

8.5 render_stream

Behaves like **render** but returns content file after conversion.

Example with Rails

```
def my_action
  doc=RGhost::Document.new
  #do something
  send_data doc.render_stream(:pdf), :filename => "/tmp/myReport.pdf"
end
```

8.5.1 TCP/IP direct printer

```
require 'socket'
```

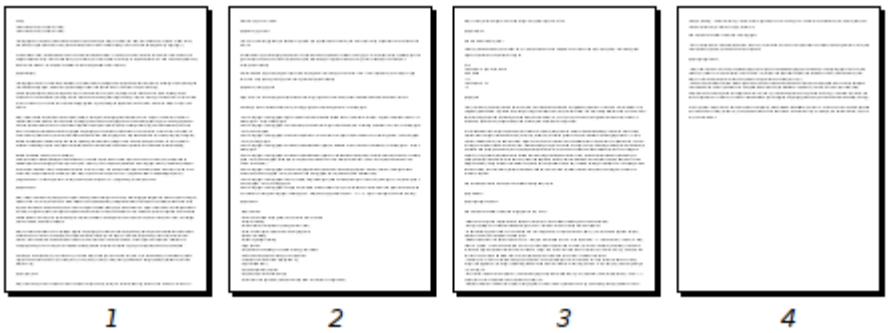
```
require 'rghost'


doc=Document.new
 #do something
printer = TCPSocket.open('192.168.1.70', 9100)
printer.write doc.render_stream(:ps)
printer.close
```

8.6 Virtual pages

With method **virtual_pages** you can **define** any number of virtual pages per physical page. The cursor on the virtual page jumps in columns for each virtual page and runs primitive next_page when columns end. Look at the example below. Example for a document without virtual pages. We will have

```
doc=Document.new
doc.text File.readlines("/tmp/mytext.txt")
```

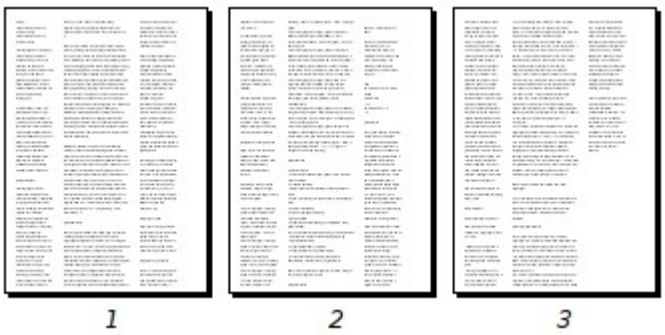
will generate



Now for a document with 3 virtual pages

```
doc=Document.new
doc.virtual_pages do
  new_page :width => 4
  new_page :width => 7, :margin_left => 1
  new_page :width => 4, :margin_left => 1
end
doc.text File.readlines("/tmp/mytext.txt")
```

will generate



PS: The parameter margin_left on the first virtual page won't be used because it will use the page's margin_left.

9. Graphic shapes

9.1 Border

Border object renders a border around vector shapes. Its sketch can be a combination of color, dashes, line joints and line caps. Usually it's used inside object facades, such as Document, CallbackFacade and PsFacade as parameter :border, for example:

```
d=Document.new
d.horizontal_line :middle, :border => { :color => '#058412', :dash => [1,0,2] }
```

You can use it as a new instance of Border and set it inside Document by method **set,** example:

```
d=Document.new
b=Border.new :color => '#058412', :dash => [1,0,2]
d.set b
d.lineto :x => 2.5, :y => 5
```

Options

:color - Facade to **Color** class using the same parameters.
:dash - Facade to **Dash** class using the same parameters.
:width - Facade to **LineWidth class** using the same parameter.
:linejoin - Sets the line join parameter in the graphics state to int, which must be one of the integers 0, 1, or 2.

9.1.1 :linejoin examples

Miter join

```
:linejoin => 0
```

Round join

```
:linejoin => 1
```

Bevel join

```
:linejoin => 2
```

9.1.1 :linecap examples

:linecap - Sets the line cap parameter in the graphics state to int, which must be one of the integers 0, 1, or 2
Butt cap

```
:linecap => 0
```

Round cap

```
:linecap => 1
```

Projecting square cap

```
:linecap => 2
```

9.2 Color

The method create is a color factory depending on the parameter used. The parameter varies between 0 and 1, if value is greater that 1, it will be divided by 100.0 . The class Color Options :color is normally used in passing parameters in the construction of objects via facade method.

9.2.1 Creating RGB color

String HTML color converter

```
Color.create '#FFAA33'
```

Using Symbol class, the **create** method will be found in RGhost::Constants::Color::RGB

```
Color.create :red
```

Using Array with 3 elements

```
Color.create [0.5, 0.3, 0.5]
```

Hash with 3 pairs of key/value. Valid keys :red, :green and :blue

```
Color.create :red => 0.5, :green => 0.3, :blue => 0.5
```

Hash with 3 pairs of key/value. Valid keys :r, :g and :b

```
Color.create :r => 0.5, :g => 0.3, :b => 0.5
```

9.2.2 Creating CMYK color

Hash with 4 pair of key/value. Valids keys :cyan, :magenta, :yellow and :black

```
 Color.create :cyan=> 1 ,:magenta => 0.3, :yellow => 0, :black => 0
```

Hash with 4 pair of key/value. Valids keys :c, :m, :y and :b

```
 Color.create :c=> 1 ,:m => 0.3, :y => 0, :b => 0
```

9.2.3 Creating Gray color

A single Numeric

```
Color.create 0.5
```

50 percent of black will be divided by 100.0

```
Color.create 50
```

9.3 Dash

Sets the dash pattern on border lines. It reads one array all of which must be non-negative numbers and not all zero. It behaves this way

```
[2,1]     #=> 2 turn on and 1 off, 2 turn on and 1 off ...
[3,1,2,5] #=> 3 on, 1 off, 2 on, 5 off ... repeating until end
```

Examples using dash as border parameter

```
d=Document.new
d.horizontal_line(:bottom, :border=>{:dash => 1, :width => 2 })
```



```
d=Document.new
d.horizontal_line(:bottom, :border=>{:dash => [1,1], :width => 2 })
```



```
d=Document.new
d.horizontal_line(:bottom, :border=>{:dash => [1,2,1], :width => 2 })
```



```
d=Document.new
d.horizontal_line(:bottom, :border=>{:dash => [2,10,5], :width => 2 })
```



```
d=Document.new
d.horizontal_line(:bottom, :border=>{:dash => [1,1,3,1,5,1,7,1,9,1,10], :width => 4 })
```


9.3.1 Examples using Dash class

```
d=Document.new
d.scale(1,8)
d.set Dash.new([1,1,2,1,2,1,3])
d.line_width 3
d.lineto :x => :limit_right, :y => :Y
```

9.3 Line Width

Sets the line width parameter in the graphics state. Examples The thinner line

```
doc=Document.new
doc.line_width 0
```



```
doc.line_width 1
```



```
doc.line_width 2
```


9.4 Line

Draws line from one point to another; the first point is created by moveto and the last point is created using the method lineto.

Examples

```
doc=Document.new
doc.moveto :x => 2, :y => 3
doc.lineto :x => 5, :y => 2
```

```
doc=Document.new
doc.moveto :x => 2, :y => 3
doc.lineto :x => 4, :y => 4
```

```
doc=Document.new
doc.border :color => '#AAFA49', :width => 4
doc.moveto :x => 2, :y => 3
doc.lineto :x => 4, :y => 1
```



```
doc=Document.new
doc.border :color => '#49AAFA', :width => 1
doc.moveto :x => 2, :y => 3
doc.lineto :x => 4, :y => 1
doc.moveto :x => 2, :y => 3
doc.lineto :x => 2, :y => 1
```

Using graphic state to close path shape

```
doc=Document.new
doc.graphic do
  doc.border :color => '#49AAFA', :width => 1
  doc.moveto :x => 2, :y => 3
  doc.lineto :x => 4, :y => 1
  doc.lineto :x => 2, :y => 1
  doc.shape_content :color => "#F0FFFF"
  doc.closepath
end
```

9.4.1 Horizontal Line

Creates horizontal line on the current row.

Examples

Drawing line in the middle

```
doc.show "Foo Bar"
doc.horizontal_line :middle
```


Drawing line on the bottom and customizing border attributes

```
doc.show "Foo Bar"
doc.horizontal_line :bottom, :border => {:dash => [1,2,2,2], :color => :red}
```

Specifies size and where the line will begin

```
 doc.show "Foo Bar"
 doc.horizontal_line :top, :start_in => 2, :size => 5, :border => {:dash => [1,2,2,2],
:color => :red}
```

9.4.2 Vertical Line

Draws a vertical line where :start_in => y position and :size => size of line.

Example

```
doc=Document.new
doc.vertical_line :start_in => 1, :size => 2, :border => {:color => :red}
```

9.4.2.1 Vertical Line row

Draws a vertical line with size or :row_height of the document.

```
doc=Document.new
doc.vertical_line_row
```

9.5 Frame

Creates one rectangle or one shape with rounded corners.

Options

`:x` and **`:y`** - Coordinates for position.
`:corners` - Value for rounded corners. Use 0 for straight angle.
`:width` and **`:height`** - Size of frame
`:content` - facade to **ShapeContent** with same parameters.
`:border` - facade to **Border** with same parameters.

Examples using facade frame method inside Document.

```
d=Document.new
d.frame :x => 3, :width => 7, :height => 5, :content => {:fill => false}
```

```
d=Document.new
d.frame :x => 3, :width => 7, :height => 5, :content => {:color => '#35F6A3' }
```

```
d=Document.new
d.frame :x => 3, :width => 7, :height => 5, :content => {:color => '#35F6A3' }, :border
=>{:width => 5, :dash => [1,3,10]}
```

```
d=Document.new

d.frame :x => 3, :width => 7, :height => 5, :content => {:color => '#35F6A3' }, :corners =>
20
```

```
d=Document.new

d.frame :x => 3, :width => 7, :height => 5, :content => {:color => :yellow }, :border =>
{:color => :red, :width => 4}, :corners => 20
```

9.6 Polygon

Draws one shape based on relative node points.

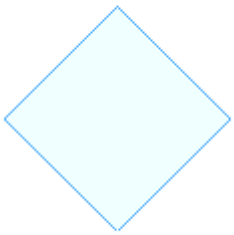
Options

`:x` and **`:y`** - Initial position.
`:content` - Facade to **ShapeContent** with same parameters.
`:border` - Facade to **Border** with same parameters.

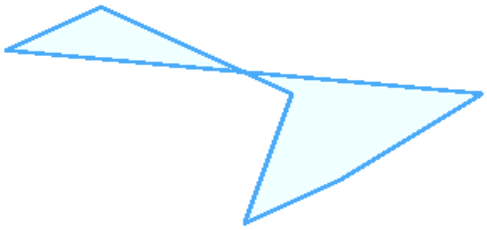
Examples

```
doc.polygon :x => 3.5, :y => 5.5 do
  node :x => 4,  :y => 0
  node :x => 0,  :y => -4
  node :x => -4, :y => 0
  node :x => 0,  :y => 4
end
```

```
doc.polygon :x => 3.5, :y => 4.5 do
  node :x => 2,  :y => 2
  node :x => 2,  :y => -2
  node :x => -2, :y => -2
end
```



```
doc.polygon :x => 1, :y => 5, :border => {:width => 2, :linejoin => 1} do
  node :x => 2,  :y => 2/2
  node :x => 2*2,:y => -2
  node :x => -1, :y => -3
  node :x => 2,  :y => 1
  node :x => 3,  :y => 2
end
```



9.7 Circle

Draws a circle to the current point (or current row by default).

Options

`:x` and **`:y`** - as center of the circle.
`:radius` - as radius(in points).
`:ang1` - the angle of a vector from (:x , :y) of length :radius to the first endpoint of the circle.
`:ang2` - the angle of a vector from (:x, :y) of length :radius to the second endpoint of the circle.
`:content` - facade to **ShapeConent** with same parameters.
`:border` - facade to **Border** with same parameters.
`:use` - **`:arc`** draws counterclockwise and **`:arcn`** (arc negative) clockwise direction.

Examples using facade circle method inside Document.

```
d=Document.new
d.circle :x => 5, :y => 2.5 , :radius => 40
```

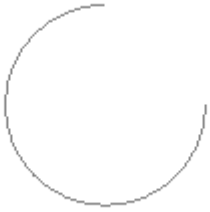
```
d=Document.new
d.circle :x => 5, :y => 2.5 , :radius => 40, :content => {:fill => false}
```

```
d=Document.new
d.circle :x => 5, :y => 2.5 , :radius => 40, :content => {:color => "#FF0000"}
```

```
d=Document.new
d.circle :x => 5, :y => 2.5 , :radius => 40, :content => {:color => "#FF0000"} ,:border =>
{:color => "#FFFFFF"}
```

```
d=Document.new
d.circle :x => 5, :y => 2.5 , :radius => 40, :content => {:color => :yellow} ,:border =>
{:color => :orange, :dash => [1,2,1,2], :width => 20}
```

```
  d=Document.new
  colors=%w[#98AE09 #AFE099 #A971FF #CC1010 #FF7201 #34FEE1]
  6.downto(1) do |v|
    d.circle :x => 5, :y => 2.5 , :radius => v*10, :content =>{:color => colors[v]}
  end
```



```
d=Document.new
d.circle :x => 5, :y => 2.5 , :ang1 => 90, :radius => 50, :content => {:fill => false }
```

```
d=Document.new
d.circle :x => 5, :y => 2.5 , :ang2 => 90, :radius => 50, :content => {:fill => false }
```

```
d=Document.new
d.circle :x => 5, :y => 2.5 , :ang2 => 90, :radius => 50, :content =>{:color => :green}
```

```
 d=Document.new
 d.circle :x => 5, :y => 2.5 , :ang2 => 90, :use => :arcn, :radius => 50, :content =>{:color
=> :green}
```

Examples using Circle class

```
  d=Document.new
  d.scale(3,1)
  d.set Circle.new(:x => 1.5, :y => 1.5 , :ang2 => 180, :radius => 25)
```

9.8 Default properties

You can define your own properties by replacing the default value of the classes Border and ShapeContent, like this below

```
RGhost::Border::DEFAULT_OPTIONS[:color] = '#FF3366'
RGhost::ShapeContent::DEFAULT_OPTIONS[:color] = '#AA1134'
```

10. Data grids

This is a helper object to create a tabular representation composed of rows, columns and title.

10.1 Simple array manipulation

This prototype is used to join the common attributes for data grids. To use a Grid you must first set up the columns , then load the data. Example:

```
grid=Grid::Matrix.new
grid.column :title => "Code", :width => 1
grid.column :title => "Name", :width => 3, :align => :center
grid.column :title => "Date", :width => 3, :align => :center
```

Note that both the width and alignment of the last two columns are identical. To avoid this, you can specify a default width by passing the values to the Grid::Matrix object constructor, so any columns that do not specify these values will use them. Example:

```
grid=Grid::Matrix.new :width => 3, :align => :center
grid.column :title => "Code", :width => 1    #:width => 1, :align => :center
grid.column :title => "Name"                 #:width => 3, :align => :center
grid.column :title => "Date"                 #:width => 3, :align => :center
```

The actual content needs to be passed in as an array of arrays

```
values=[
  [1,"Name 1", Time.now],
  [2,"Name 2", Time.now],
  [3,"Name 3", Time.now],
  [4,"Name 4", Time.now],
  [5,"Name 5", Time.now]
]
```

Bind the content to the grid:

```
grid.data(values)
```

Add the Grid to a document

```
d=Document.new
d.set grid
```

10.2 Importing CSV

Grid::CSV allows you to import data from a csv file.
Example

```
grid=Grid::CSV.new :width => 2
grid.column :title => "User", :align => :right
grid.column :title => "Password", :format => lambda{|v| (v.to_s == "x") ? "Yes" : "No"}
grid.column :title => "UID", :width => 1
grid.column :title => "GID"
grid.column :title => "Gecos", :width => 2.5
grid.column :title => "Home Dir", :width => 4
grid.column :title => "Shell"
grid.style  :bottom_lines
grid.data("/etc/passwd",/:/)
```

result

User	Password	UID	GID	Gecos	Home Dir	Shell
root	Yes	0	0	root	/root	/bin/bash
shairon	Yes	0	0	Shairon Toledo,,,	/local/shairon	/bin/bash
daemon	Yes	1	1	daemon	/usr/sbin	/bin/sh
bin	Yes	2	2	bin	/bin	/bin/sh
sys	Yes	3	3	sys	/dev	/bin/sh
sync	Yes	4	65534	sync	/bin	/bin/sync
games	Yes	5	60	games	/usr/games	/bin/sh
man	Yes	6	12	man	/var/cache/man	/bin/sh
lp	Yes	7	7	lp	/var/spool/lpd	/bin/sh
mail	Yes	8	8	mail	/var/mail	/bin/sh
news	Yes	9	9	news	/var/spool/news	/bin/sh
uucp	Yes	10	10	uucp	/var/spool/uucp	/bin/sh
proxy	Yes	13	13	proxy	/bin	/bin/sh
www-data	Yes	33	33	www-data	/var/www	/bin/sh
backup	Yes	34	34	backup	/var/backups	/bin/sh
list	Yes	38	38	Mailing List Manager	/var/list	/bin/sh
irc	Yes	39	39	ircd	/var/run/ircd	/bin/sh
nobody	Yes	65534	65534	nobody	/nonexistent	/bin/sh
libuuid	Yes	100	101		/var/lib/libuuid	/bin/sh
dhcp	Yes	101	102		/nonexistent	/bin/false
syslog	Yes	102	103		/home/syslog	/bin/false
klog	Yes	103	104		/home/klog	/bin/false
hplip	Yes	104	7	HPLIP system user,,,	/var/run/hplip	/bin/false
avahi-autoipd	Yes	105	113	Avahi autoip daemon,,,	/var/lib/avahi-autoipd	/bin/false
gdm	Yes	106	114	Gnome Display Manager	/var/lib/gdm	/bin/false
pulse	Yes	107	116	PulseAudio daemon,,,	/var/run/pulse	/bin/false
messagebus	Yes	108	119		/var/run/dbus	/bin/false
avahi	Yes	109	120	Avahi mDNS daemon,,,	/var/run/avahi-daemon	/bin/false
polkituser	Yes	110	122	PolicyKit,,,	/var/run/PolicyKit	/bin/false
haldaemon	Yes	111	123	Hardware abstraction layer,,,	/var/run/hald	/bin/false
shairon	Yes	1000	1000	Shairon Toledo,,,	/local/shairon	/bin/bash
postgres	Yes	112	124	PostgreSQL administrator,,,	/var/lib/postgresql	/bin/bash
dictd	Yes	113	125		/var/lib/dictd	/bin/false
sshd	Yes	114	65534		/var/run/sshd	/usr/sbin/nologin

10.3 Active Record and Rails

Grid::Rails differs from the other grids only in the mapping of the model attributes to be printed in the column. For example, the class **Calls** inherits ActiveRecord::Base.

Example

```
rails_grid=Grid::Rails.new(:align => :center, :width => 3)
```

Map the attribute name then the already known options

```
rails_grid.column :mobile,   :title => "Mobile", :align => :right
rails_grid.column :duration, :title => "Duration",:width => 3
rails_grid.column :start,    :title => "Start", :format => :eurodate
rails_grid.column :called,   :title => "Number", :align => :right, :title_align => :center
```

Loading data

```
calls = Call.find(:all, :limit => 5000)
rails_grid.data(calls)
```

Relationships

For relationships we use a block or a String instead of an attribute name (Symbol). Example

```
class Accounts < ActiveRecord::Base
  belongs_to :customer
end
class Customers < ActiveRecord::Base
  has_many :accounts
end
c = Customers.find(:all, :include => :accounts )
```

Using a block

```
g=Grid::Rails.new  :width => 4
g.column :id,         :title => "Customer id"
g.column :name,       :title => "Name",   :width => 6
g.column :created_on, :title => "Date ",  :format => lambda{|d| d.strftime("%m/%d/%Y") }
g.column lambda { accounts.first.login }, :title => "Login"
g.column lambda { accounts.first.type },  :title => "Account Type"
g.data(c)
```

Or a String

```
g.column 'accounts.first.login', :title => Login"
```

10.4 Field Format

Defines the format of data using :format parameters option.

:format - Format of the data. You can format data in four different ways with RGhost, by passing in a Symbol, a String, a Class or Proc.

:format Parameters type

Symbol - Searches for a method defined as Grid::FieldFormat::*method_name*

```
:format => :eurodate
```

Class - A class that inherits Grid::FieldFormat::Custom with an overridden format method.

```
:format => MyFormat
```

String - Formats using the same parameters used in sprintf

```
:format => "%0.2f"
```

Proc - A block. In the example a text limited to 9 characters.

```
:format => lambda {|s| s.gsub(/^(.{9}).*$/,'\1...')}
```

10.4.1 Customizing formats

Let's create a class that replaces empty spaces with a double dash.

```
class MyFormat < DataGrid::FieldFormat::Custom
  def format
    @value.to_s.gsub(/ /,'--')
  end
end
```

Using

```
grid.column :title => "Name", :format => MyFormat
```

Below, the columns with their proper formats.

```
grid.column :title => "Code",:format => "(%d)", :width => 1
grid.column :title => "Name",  :format => MyFormat
grid.column :title => "Date",  :format => lambda {|date| date.strftime("%d/%m/%Y") }
values=[
[1,"Name 1", Time.now],
[2,"Name 2", Time.now],
[3,"Name 3", Time.now],
[4,"Name 4", Time.now],
[5,"Name 5", Time.now]
]
grid.data(values)
```

Add the Grid to a document

```
d=Document.new
d.set grid
```

Id	Name	Date
(1)	Name--1	19/05/2008
(2)	Name--2	19/05/2008
(3)	Name--3	19/05/2008
(4)	Name--4	19/05/2008
(5)	Name--5	19/05/2008

10.5 Preset Styles

Grid has 3 preset styles :bottom_lines, :border_lines and old_forms. To set any of them, use:

```
 grid.style(:border_lines)
```

:border_lines - instance of Grid::Style::BorderLines

Id	Name	Date
1	Name 1	19/05/2008
2	Name 2	19/05/2008
3	Name 3	19/05/2008
4	Name 4	19/05/2008
5	Name 5	19/05/2008

:bottom_lines - instance of Grid::Style::BottomLines

Id	Name	Date
1	Name 1	19/05/2008
2	Name 2	19/05/2008
3	Name 3	19/05/2008
4	Name 4	19/05/2008
5	Name 5	19/05/2008

:old_forms - instance of Grid::Style::OldForms

Id	Name	Date
1	Name 1	19/05/2008
2	Name 2	19/05/2008
3	Name 3	19/05/2008
4	Name 4	19/05/2008
5	Name 5	19/05/2008

10.6 Grid Callbacks

The callbacks for the grid are defined here. Let's see them in action.

```
Grid::CallbackFacade examples
grid=Grid::Matrix.new :column_padding => 1
grid.column :title => "Id", :width => 1
grid.column :title => "Name", :width => 3, :align => :center
grid.column :title => "Date", :width => 3, :align => :right,
            :title_align => :center, :format => lambda{|v| v.strftime("%d/%m/%Y")}
values=('A'..'E').to_a.map{|v|  [v,"Name #{v}", Time.now]}
```

Use even_row:

```
grid.even_row do
 background_row(:size => grid.width)
end
```

Id	Name	Date
A	Name A	19/05/2008
B	Name B	19/05/2008
C	Name C	19/05/2008
D	Name D	19/05/2008
E	Name E	19/05/2008

Now use **before_row** to create a top and bottom line:

```
grid.before_row do
 horizontal_line(:top,    :size => grid.width )
 horizontal_line(:bottom, :size => grid.width)
end
```

Id	Name	Date
A	Name A	19/05/2008
B	Name B	19/05/2008
C	Name C	19/05/2008
D	Name D	19/05/2008
E	Name E	19/05/2008

before_column:

```
grid.before_column do
 vertical_line_row
end
```

Id	Name	Date
A	Name A	19/05/2008
B	Name B	19/05/2008
C	Name C	19/05/2008
D	Name D	19/05/2008
E	Name E	19/05/2008

after_column

```
grid.after_column {  vertical_line_row  }
```

Id	Name	Date
A	Name A	19/05/2008
B	Name B	19/05/2008
C	Name C	19/05/2008
D	Name D	19/05/2008
E	Name E	19/05/2008

Modifying the header

```
grid.header.before_create do
   horizontal_line(:top, :size => grid.width)
end
```

Id	Name	Date
A	Name A	19/05/2008
B	Name B	19/05/2008
C	Name C	19/05/2008
D	Name D	19/05/2008
E	Name E	19/05/2008

Finishing the grid lines:

```
grid.header.before_column do
  vertical_line_row
end

grid.header.after_column do
 vertical_line_row
end
```

Id	Name	Date
A	Name A	19/05/2008
B	Name B	19/05/2008
C	Name C	19/05/2008
D	Name D	19/05/2008
E	Name E	19/05/2008

Now, adding a bold font to the header

```
grid.header.before_create do
  horizontal_line(:top, :size => grid.width)
  use_tag :bold
 end
```

Id	**Name**	**Date**
A	**Name A**	**19/05/2008**
B	**Name B**	**19/05/2008**
C	**Name C**	**19/05/2008**
D	**Name D**	**19/05/2008**
E	**Name E**	**19/05/2008**

Oops. Not quite what we expected, since the entire grid used a bold font. We can use the header callback after_create to reset the font.

```
grid.header.after_create do
   use_tag :normal
 end
```

Id	**Name**	**Date**
A	Name A	19/05/2008
B	Name B	19/05/2008
C	Name C	19/05/2008
D	Name D	19/05/2008
E	Name E	19/05/2008

Don‘t forget

```
doc=Document.new
doc.set grid
```

11. Converting PDF into other formats

RGhost has a “utility“ class for converting PDF documents to other formats. The options from *Document#render and RGhost::Engine#render* are used for the **Convert#to** method. Example:

```
preview = RGhost::Convert.new('/my/dir/file.pdf')
preview.to :png, :multipage => true, :resolution => 72  #returns an Array of Files
```

The :range option can be used to convert only a subset of a multipage file. Example:

```
pages = Convert.new('/tmp/test.pdf').to :jpeg, :multipage => true, :range => 5..6 #returns
an Array of Files
```

Only the first page

```
thumb = Convert.new('/tmp/test.pdf').to :png  #returns File
```

Checking for errors

```
thumb = Convert.new('/tmp/test.pdf')
out = thumb.to :png  #returns File
if thumb.error
  raise Exception.new “Image error”
else
  File.cp(out,”/my/dir”)
end
```

12. Postscript API's Internal Stack

12.1 Public variables

Ruby Symbol	*Postscript Type*	*Access*	*Description*
:row	integer	R	Number of the current row
:current_row	integer	R	Y coordinate of the row
:row_height	float(real)	RW	Row height
:row_padding	float(real)	RW	Padding between rows
:limit_top	float(real)	RW	page's top minus :margin_top
:limit_right	float(real)	RW	page's right side minus :margin_right
:limit_bottom	float(real)	RW	Page's bottom minus :margin_bottom
:limit_left	float(real)	RW	Page's left minus :margin_left
:area_x	float(real)	RW	Area between :margin_left and :margin_right
:area_y	float(real)	RW	Area between :margin_top e :margin_bottom
:fontsize	integer	RW	Default font size
:font_normal	dict(font)	RW	normal
:font_bold	dict(font)	RW	Bold font
:font_italic	dict(font)	RW	Italic font
:font_bold_italic	dict(font)	RW	font bold **and** italic
:col_padding	float	RW	Column padding (blank space between the columns)
:col	integer	R	Current column
:headings?	bool	RW	returns true if it's a header
:current_table_row	integer	R	Table's current line relative to the document.
:current_page	integer	R	Current page number
:count_pages	integer	RW	Total pages
:X	{} proc => float/integer	R	current position X axis
:Y	{} proc => float | integer	R	current position Y axis
:pagesize	Array(integer)	RW	Page size. i.e.: [400,300]