PHP Pdf Class

Native PDF document creation with PHP 5.X

released under the terms of the MIT license

Contributors

Version 0.12.41

FORK ON GITHUB.COM

R&OS

https://github.com/rospdf/

Table of Contents

Introduction 1
Changelog 1
Use 3
Extensions 3
EZPDF Class Extension 4
- Cezpdf 4
- ezSetMargins 5
- ezSetCmMargins 5
- ezNewPage 5
- ezColumnsStart 5
- ezColumnsStop 5
- ezInsertMode 6
- ezSetY 6
- ezSetDy 6
- ezStartPageNumbers 6
- ezWhatPageNumber 8
- ezGetCurrentPageNumber 8
- ezStopPageNumbers 8
- ezTable 9
- ezText 10
- ezImage 11
- ezOutput 12
- ezStream 12
- Inline codes 13

Base Class Functions 15
- addText 15
- setColor 15
- setStrokeColor 15
- setLineStyle 15
- line 16
- curve 16
- ellipse 16
- partEllipse 17
- polygon 17
- rectangle 18
- filledRectangle 18
- newPage 18
- getFirstPageId 19
- stream 19
- getFontHeight 19
- getFontDescender 19
- getTextWidth 19
- saveState 19
- restoreState 19
- openObject 19

reopenObject 20
closeObject 20
addObject 20
stopObject 20
addInfo 20
setPreferences 20
addImage 21
addJpegFromFile 21
addPngFromFile 21
output 21
openHere 21
selectFont 22
setFontFamily 22
setEncryption 23
addLink 24
addInternalLink 24
addDestination 24
transaction 24
Misc 25
Callback functions 25
Units 27

Introduction

This class is designed to provide a **non-module**, non-commercial alternative to dynamically creating pdf documents from within PHP. Obviously this will not be quite as quick as the module alternatives, but it is surprisingly fast, this demonstration page is almost a worst case due to the large number of fonts which are displayed. There are a number of features which can be within a Pdf document that it is not at the moment possible to use with this class, but I feel that it is useful enough to be released.

This document describes the possible useful calls to the class, the readme.php file (which will create this pdf) should be sufficient as an introduction.

Note that this document was generated using the demo script 'readme.php' which came with this package.

Changelog

0.12.41

- fixed issue #80
- applied several PR's related to coding standards
- replaced pack with md5 raw output

0.12.40

- make addImage method public #73 - example imagetext.php

0.12.39

- fixed issue #71
- reduce callbacks proberly when reattaching in addText
- improved stability and performance

0.12.38

- finally fixed special char issue when encrypting documents (document info - #70)

0.12.37

- fixed issue #70

0.12.36

- released under the MIt license
- fixed issue #55

Please refer to https://github.com/rospdf/pdf-php for all previous changes

Use

It is free for use for any purpose (public domain), though we would prefer it if you retain the notes at the top of the class containing the authorship, and feedback details.

Note that there is no guarantee or warranty, implied or otherwise with this class, or the extension.

Extensions

In order to create simple documents with ease a class extension called 'ezPdf' has been developed, at the moment this includes auto page numbering, tabular data presentation, text wrapping to new pages, etc. This document has been created using mostly ezPdf functions.

The functions of ezpdf are described in the next section, though as it is a class extension, all of the basic functions from the original class are still available.

EZPDF Class Extension

(note that the creation of this document in readme.php was converted to ezpdf with the saving of many lines of code).

It is anticipated that in practise only the simplest of documents will be able to be created using just ezpdf functions, they should be used in conjunction with the base class functions to achieve the desired result.

Cezpdf ([paper='a4'],[orientation='portrait'],[type='none'],[options=array()])

This is the constructor function, and allows the user to set up a basic page without having to know exactly how many units across and up the page is going to be.
Valid values for paper are listed below, a two or four member array can also be passed, a two member array will be interpreted as the size of the page in centimeters, and a four member array will be the size of the page in points, similar to the call to the base calss constructor function.
Starting ezpdf with the code below will create an a4 portrait document.

$paper The valid values for the paper (thanks to the work of Nicola Asuni) are:
'4A0', '2A0', 'A0', 'A1', 'A2', 'A3', 'A4', 'A5', 'A6' , 'A7', 'A8', 'A9', 'A10', 'B0', 'B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'B8', 'B9', 'B10', 'C0', 'C1', 'C2' , 'C3', 'C4' , 'C5', 'C6', 'C7', 'C8', 'C9', 'C10' , 'RA0', 'RA1', 'RA2', 'RA3', 'RA4', 'SRA0', 'SRA1', 'SRA2', 'SRA3', 'SRA4', 'LETTER', 'LEGAL', 'EXECUTIVE', 'FOLIO'

$orientation 'portrait' and 'landscape' can be used

$type following types are possible: 'none' | 'color' | 'image'

$options if type is diffent then 'none' $options can be set as follows
if $type is set to 'color'
$options['r'] = red-component of backgroundcolour (0 <= r <= 1)
$options['g'] = green-component of backgroundcolour (0 <= g <= 1)
$options['b'] = blue-component of backgroundcolour (0 <= b <= 1)
if $type is set to 'image' then the $options array has other attributes
$options['img'] = location of image file; URI's are allowed if allow_url_open is enabled
$options['width'] = width of background image; default is width of page
$options['height'] = height of background image; default is height of page
$options['xpos'] = horizontal position of background image; default is 0
$options['ypos'] = vertical position of background image; default is 0
$options['repeat'] = repeat image horizontally (1), repeat image vertically (2) or full in both directions (3); default is 0
highly recommend to set this->hashed to true when using repeat function

This document shows you the 'color' type with following:

```
$pdf = new Cezpdf('a4','portrait','color',array(0.8,0.8,0.8));
```

If you want to get started in an easy manner, then here is the 'hello world' program:

```
<?php
include ('class.ezpdf.php');
$pdf = new Cezpdf();
$pdf->selectFont('Helvetica');
$pdf->ezText('Hello World!',50);
$pdf->ezStream();
?>
```

ezSetMargins (top,bottom,left,right)

Sets the margins for the document, this command is optional and they will all be set to 30 by default. Setting these margins does not stop you writing outside them using the base class functions, but the ezpdf functions will wrap onto a new page when they hit the bottom margin, and will not write over the side margins when using the **ezText** command below.

ezSetCmMargins (top,bottom,left,right)

Sets the margins for the document using centimeters

ezNewPage ()

Starts a new page. This is subtly different to the newPage command in the base class as it also places the ezpdf writing pointer back to the top of the page. So if you are using the ezpdf text functions, then this is the one to use when manually requesting a new page.

ezColumnsStart ([options])

This will start the text flowing into columns, *options* is an array which contains the control options.

The options are:
'gap' => the gap between the columns
'num' => the number of columns.

Both options (and the array itself) are optional, if missed out then the defaults are gap=10, num=2;

Example calls could be (you would use only one of these):

```
$pdf->ezColumnsStart();
$pdf->ezColumnsStart(array('num'=>3));
$pdf->ezColumnsStart(array('num'=>3,'gap'=>2));
$pdf->ezColumnsStart(array('gap'=>20));
```

ezColumnStop is used to stop multi-column mode.

ezColumnsStop

This stops multi-column mode, it will leave the writing point at whatever level it was at, it is recommended that an ezNewPage() command is executed straight after this command, but

for flexibility this is left up to the individual consumer.

ezInsertMode ([status=1,$pageNum=1,$pos='before'])

This command can be used to stop and start page insert mode, while this mode is on then any new page will be inserted into the midst of the document. If it is called with status=1, then insert mode is started and subsequent pages are added before or after page number 'pageNum'. 'pos' can be set to 'before' or 'after' to define where the pages are put. The 'pageNum' is set to which page number this position is relative to.

All subsequent pages added with ezNewPage are then inserted within the document following the last inserted page.

Insertion page is ended by calling this command with 'status'=0, and further pages are added to the end of the document.

ezSetY (y)

Positions the ezpdf writing pointer to a particular height on the page, don't forget that pdf documents have **y-coordinates which are zero at the bottom of the page and increase as they go up** the page.

ezSetDy (dy [,mod])

Changes the vertical position of the writing point by a set amount, so to move the pointer 10 units down the page (making a gap in the writing), use:

```
ezSetDy(-10)
```

If this movement makes the writing location below the bottom margin, then a new page will automatically be made, and the pointer moved to the top of it.

The optional parameter 'mod' can be set to the value 'makeSpace', which means that if a new page is forced, then the pointer will be moved the distance 'dy' on the new page as well. The intention of this is if you needed 100 units of space to draw a picture, then doing:

```
ezSetDy(-100,'makeSpace')
```

guarantees that there will be 100 units of space above the final writing point.

ezStartPageNumbers (x,y,size,[pos],[pattern],[num]) = setNum

Add page numbers on the pages from here, place then on the 'pos' side of the coordinates (x,y) (pos can be 'left' or 'right').
Use the given 'pattern' for display, where {PAGENUM} and {TOTALPAGENUM} are replaced as required, by default the pattern is set to '{PAGENUM} of {TOTALPAGENUM}'
If $num is set, then make the first page this number, the number of total pages will be adjusted to account for this.

the following code produces a seven page document, numbered from the second page

(which will be labelled '1 of 6'), and numbered until the 6th page (labelled '5 of 6')

```
$pdf = new Cezpdf();
$pdf->selectFont('Helvetica');
$pdf->ezNewPage();
$pdf->ezStartPageNumbers(300,500,20,'','',1);
$pdf->ezNewPage();
$pdf->ezNewPage();
$pdf->line(300,400,300,600); // line drawn to check 'pos' is working
$pdf->ezNewPage();
$pdf->ezNewPage();
$pdf->ezNewPage();
$pdf->ezStopPageNumbers();
$pdf->ezStream();
```

This function was modified in version 009 to return a page numbering set number (setNum in the call above), this allows independent numbering schemes to be started and stopped within a single document. This number is passed back into *ezStopPageNumbers* when it is called.

Here is a more complex example:

```
$pdf->selectFont('Helvetica');
$pdf->ezNewPage();
$i=$pdf->ezStartPageNumbers(300,500,20,'','',1);
$pdf->ezNewPage();
$pdf->ezNewPage();
$pdf->ezStopPageNumbers(1,1,$i);
$pdf->ezNewPage();
$i=$pdf->ezStartPageNumbers(300,500,20,'','',1);
$pdf->ezNewPage();
$pdf->ezNewPage();
$pdf->ezStopPageNumbers(1,1,$i);
$pdf->ezNewPage();
$i=$pdf->ezStartPageNumbers(300,500,20,'','',1);
$pdf->ezNewPage();
$pdf->ezNewPage();
$pdf->setColor(1,0,0);
$pdf->ezNewPage();
$pdf->ezStopPageNumbers(1,1,$i);
$pdf->ezNewPage();
$i=$pdf->ezStartPageNumbers(300,500,20,'','',1);
$pdf->ezNewPage();
$pdf->ezNewPage();
$pdf->ezNewPage();
$pdf->ezNewPage();
$j=$pdf->ezStartPageNumbers(300,400,20,'','',1);
$k=$pdf->ezStartPageNumbers(300,300,20,'','',1);
$pdf->ezNewPage();
$pdf->ezNewPage();
$pdf->ezNewPage();
$pdf->ezStopPageNumbers(1,1,$i);
$pdf->ezNewPage();
$pdf->ezNewPage();
$pdf->ezStopPageNumbers(1,1,$j);
$pdf->ezStopPageNumbers(0,1,$k);
$pdf->ezNewPage();
$pdf->ezNewPage();
```

This will create a document with 23 pages, the numbering shown on each of the pages is:

page	contents
1	blank
2	1 of 3
3	2 of 3
4	3 of 3
5	1 of 3
6	2 of 3
7	3 of 3
8	1 of 3
9	2 of 3
10	3 of 3
11	4 of 4
12	1 of 8
13	2 of 8
14	3 of 8
15	4 of 8
16	5 of 8, 1 of 6, 1 of 8
17	6 of 8, 2 of 6, 2 of 8
18	7 of 8, 3 of 6, 3 of 8
19	8 of 8, 4 of 6, 4 of 8
20	5 of 6, 5 of 8
21	6 of 6, 6 of 8
22	blank
23	blank

ezWhatPageNumber (pageNum,[setNum]) = num

Returns the number of a page within the specified page numbering system.

'pageNum' => the absolute number of the page within the document (this is based on the order that they are created).
'setNum' => the page numbering set, returned from the *ezStartPageNumbers* command.

ezGetCurrentPageNumber

return the page number of the current page

ezStopPageNumbers ([stopTotal],[next],[setNum])

In version 009 this function was enhanced to include a number of extra parameters:

'stopTotal' => 0 or 1 (default 0), stops the totaling for the page numbering set. So for example if you start numbering a 10 page document on the second page, and stop it on the 4th page, with stopTotal set to 1, then the numbers will be reported as "x of 3".
'next' => 0 or 1, stops on the next page, not this one.
'setNum' => (defaults to 0) define which set number is to be stopped, this is the number

returned from the *ezStartPageNumbers* command.

ezTable (array data,[array cols],[title],[array options]) = y

$data The easy way to throw a table of information onto the page, can be used with just the data variable, which must contain a two dimensional array of data. This function was made with data extracted from database queries in mind, so is expecting it in that format, a two dimensional array with the first array having one entry for each row (and each of those is another array).

The table will start writing from the current writing point, and will proceed until the all the data has been presented, by default, borders will be drawn, alternate limes will be shaded gray, and the table will wrap over pages, re-printing the headers at the top of each page.

The return value from the function is the y-position of the writing pointer after the table has been added to the document.

since 0.12.27 $data can contain the follow keys to colorize cells

```
'[columnName]Fill' => array(r,g,b)  // used to fill a cell background color
'[columnName]Color' => array(r,g,b) // used to color the cell text
```

The other options are described here below

$cols (optional) is an associative array, the keys are the names of the columns from $data to be presented (and in that order), the values are the titles to be given to the columns, if this is not wanted, but you do want later options then '' (the empty string) is a suitable placeholder.

$title (optional) is the title to be put on the top of the table

$options is an associative array which can contain:
'showHeadings' => 0 or 1 (enable or disable head line)
'shaded'=> 0,1,2, default is 1 (alternate lines are shaded)
0->no shading, 2->both sets are shaded
'shadeCol' => (r,g,b) array, defining the colour of the shading, default is (0.8,0.8,0.8)
'shadeCol2' => (r,g,b) array, defining the colour of the shading of the second set, default is (0.7,0.7,0.7), used when 'shaded' is set to 2.
'fontSize' => 10
'textCol' => (r,g,b) array, text colour
'titleFontSize' => 12
'rowGap' => 2 , the space between the text and the row lines on each row
'colGap' => 5 , the space between the text and the column lines in each column
'lineCol' => (r,g,b) array, defining the colour of the lines, default, black.
'xPos' => 'left','right','center','centre',or coordinate, reference coordinate in the x-direction
'xOrientation' => 'left','right','center','centre', position of the table w.r.t 'xPos'. This entry is to be used in conjunction with 'xPos' to give control over the lateral position of the table.
'width' => <number>, the exact width of the table, the cell widths will be adjusted to give the table this width.
'maxWidth' => <number>, the maximum width of the table, the cell widths will only be adjusted if the table width is going to be greater than this.
'cols' => array(<colname>=> array('justification'=>'left', 'width'=>100, 'link'=><linkColName>,

'bgcolor'=> (r,g,b)), <colname>=>....) allow the setting of other paramaters for the individual columns, each column can have its width and/or its justification set.
'innerLineThickness' => <number>, the thickness of the inner lines, defaults to 1
'outerLineThickness' => <number>, the thickness of the outer lines, defaults to 1
'protectRows' => <number>, the number of rows to keep with the heading, if there are less than this on a page, then all is moved to the next page.
'nextPageY'=> true or false (eg. 0 or 1) Sets the same Y position of the table for all future pages

0.12.9:
'shadeHeadingCol'=>(r,g,b) array, defining the backgound color of headings, default is empty

0.12.11:
'gridlines'=> EZ_GRIDLINE_* default is EZ_GRIDLINE_DEFAULT, overrides 'showLines' to provide finer control
'alignHeadings' => 'left','right','center'

0.12.27:
'evenColumns' => 0,1 make all column width the same size

Below is an example output of ezTable containing the following $data

```
$data = array(
 array('num'=>1,'name'=>'gandalf','type'=>'wizard')
,array('num'=>2,'name'=>'bilbo','type'=>'hobbit','url'=>'https://github.com
/rospdf/pdf-php')
,array('num'=>3,'name'=>'frodo','type'=>'hobbit')
,array('num'=>4,'name'=>'saruman','type'=>'bad
dude','url'=>'https://github.com/rospdf/pdf-php')
,array('num'=>5,'name'=>'sauron','type'=>'really bad dude')
);

$pdf->ezTable($data,$cols,'',array(
    'gridlines'=> EZ_GRIDLINE_DEFAULT,
    'shadeHeadingCol'=>array(0.6,0.6,0.5),
    'width'=>400,
    'cols'=> array(
        'name'=>array('bgcolor'=>array(0.9,0.9,0.7)),
        'type'=>array('bgcolor'=>array(0.6,0.4,0.2))
    )
));
```

Number	Name	Type
1	gandalf	wizard
2	bilbo	hobbit
3	frodo	hobbit
4	saruman	bad dude
5	sauron	really bad dude

Please refer to examples/tables.php for more demos

ezText (text,[size],[array options]) = y

This is designed for putting blocks of text onto the page. It will add a string of text to the document (note that the string can be very large, spanning multiple pages), starting at the

current drawing position. It will wrap to keep within the margins, including optional offsets from the left and the right, if $size is not specified, then it will be the last one used, or the default value (12 I think). The text will go to the start of the next line when a return code "\n" is found.

The return value from the function (y) is the vertical position on the page of the writing pointer, after the text has been added.

possible options are:
'left'=> number, gap to leave from the left margin
'right'=> number, gap to leave from the right margin
'aleft'=> number, absolute left position (overrides 'left')
'aright'=> number, absolute right position (overrides 'right')
'justification' => 'left','right','center','centre','full'

only set one of the next two items (leading overrides spacing)
'leading' => number, defines the total height taken by the line, independent of the font height.
'spacing' => a real number, though usually set to one of 1, 1.5, 2 (line spacing as used in word processing)

This function now supports the use of underlining markers (<u> </u>), these are implemented via a callback function, which is information that you need only of you want to use them in functions in the base class, as these markers will not work there and the full callback function markers wil have to be used (though note that ezPdf stillhas to be part of the class object as the callback function are included with that code base. The callback function markers would look like <c:uline>this</c:uline>.

ezImage (image,[padding],[width],[resize],[justification], [angle],[array border])

This function makes it much easier to simply place an image (either jpeg or png) within the flow of text within a document. Although this function can certainly be used with full page documents, its functionality is most suited to documents that are formatted in multiple columns.

This function can be used by simply supplying the name of an image file as the first argument. The image will be resized to fit centered within the current column with the default padding of 5 on each side.

The arguments are:

$image is a string containing the filename and path of the jpeg or png image you want to insert into the page. If allow_url_fopen is enabled in the PHP ini settings this can be an HTTP or FTP URL.

$padding (optional) is the number of page units that will pad the image on all sides. The default is five (5). If you do not want any padding, you may enter 0.

$width (optional) is the width of the image on the page. The default is to use the actual width of the image in pixels. Whether or not you specify a width, the actual width of the image on the page will depend on what you enter for the $resize argument as well as the

placement of the image on the page.

$resize (optional) can be one of 'full', 'width', or 'none'. The default value is 'full'.

The value 'none' means that an image will not be sized up to fit the width of a column and will not be sized down vertically to fit within the current page. If the image is too long to fit on the page it will be placed on the next page or column. If the image is too wide to fit within the current column, it will still be sized down. This is because there is no alternative, other than to actually let the image run off the page.

The value 'width' behaves the same as 'none' with the exception that images will be resized to fit the width of the current column if the given width is smaller that the current column (minus the padding).

The value 'full' behaves the same as 'width' with the exception that images will be resized down vertically to fit within the current page and column if their height is too long to fit.

$justification (optional) determines the horizontal position of the image within the current column. The default is 'center', and can be specified as either 'center', 'left', or 'right'. This setting has little meaning if the image has been resized to fit the column and only makes a practical difference if the width of the image is smaller than the width of the column.

$angle (optional) allows you the rotate the image

$border (optional) is an array which specifies the details of the border around the image. The default is no border if this argument is not given. You may specify any of the following border elements:

$border['width'] is the width of the border. The default is 1.
$border['cap'] is the cap type as specified in setLineStyle. The default is 'round'.
$border['join'] is the join type as specified in setLineStyle. The default is 'round'.
$border['color'] is an associative array for specifying the line color of the border.
The values are as specified in setStrokeColor and should be assigned to:
$border['color']['red'], $border['color']['green'] and $border['color']['blue'] respectively.

ezOutput ([debug])

Very similar to the output function from the base class, but performs any closing tasks that ezpdf requires, such as adding the page numbers.

If you are using ezpdf, then you should use this function, rather than the one from the base class.

ezStream ([options])

Very similar to the stream function from the base class (all the same options, see later in this document), but performs any closing tasks that ezpdf requires, such as adding the page numbers.

If you are using ezpdf, then you should use this function, rather than the one from the base class.

Inline codes

There are a few callback functions (see callback functions) which are contained within the ezPdf class, these are intended to make life easier. They enable complex operations to be done by including codes within the text stream.

Underline
Though underlining is supported in the ezPdf class by using the <u> directive,
if you use the base class functions to add text (such as addtext) then this wont work, instead you can use the *uline* callback function.

(Note that what the ezPdf class does internally is convert the <u> and </u> directives into callback function calls)

So as an example, this code adds some text with two pieces of underlining, one done each way

```
$pdf->ezText('The <u>quick brown</u> fox, <c:uline>is
sick of jumping</uline> the lazy dog')
```

The quick brown fox, is sick of jumping the lazy dog

Links to URLs

If you are adding links to a document, it is quite tricky to figure out where to put the rectange which will be clckable, especially if the text in teh link starts wrapping across pages, etc.

The *alink* callback allows for simple insertion of links, the format is:

<c:alink:*your_url_here*>text to be clickable</c:alink>

So as an example:

```
$pdf->ezText('<c:alink:https://github.com/rospdf/pdf-php/ pdf
class</c:alink>');
```

R&OS pdf class

Links within the document

There is a directive similar to *alink*, but designed for linking within the document, this is the *ili nk* callback function.

It is similar to *alink* except that instead of providing a URL the label of a pre-created destination should be used.

<c:ilink:*destination_label*>text to be clickable</c:ilink>

```
// place required to be marked
$pdf->addDestination('xxxyyyzzz','Fit');
// add lots of stuff, new pages etc, then...
$pdf->ezText('<c:ilink:xxxyyyzzz>R&OS pdf class</c:ilink>');
```

Click here to go to the 5th item on the table of contents.

Note that the code for the example and the actual one shown are not identical for technical reasons.

Base Class Functions

addText (x,y,size,text[,width=0][,justification='left'][,angle=0][,wordSpaceAdjust=0][,test=0)

Add the text at a particular location on the page, noting that the origin on the axes in a pdf document is in the lower left corner by default.
An angle can be supplied as this will do the obvious (in degrees).

'adjust', gives the value of units to be added to the width of each space within the text. This is used mainly to support the justification options within the ezpdf ezText function.

The text stream can now (version 006) contain directives to make the text bold and/or italic. The directives are similar the basic html:

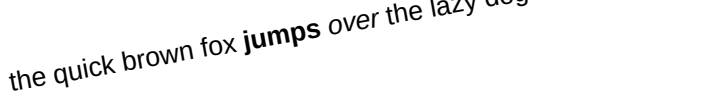
bold text
italic text
bold italic text

Note that there can be no spaces within the directives, and that they must be in lower case.

Especially **bold** and *italic* requires the style of the font being used

If you wish to print an actual '<', most of the time it would cause no problem, except in the instance where it would form a directive. Use HTML special char < and >

```
$pdf->addText(150,$y,10,"the quick brown fox <b>jumps</b>
<i>over</i> the lazy dog!",0, 'left',-10);
```



setColor (r,g,b,[force=0])

Set the fill colour to the r,g,b triplet, each in the range 0->1.
If force is set, then the entry will be forced into the pdf file, otherwise it will only be put in if it is different from the current state.

setStrokeColor (r,g,b,[force=0])

Set the stroke color, see the notes for the fill color.

setLineStyle ([width],[cap],[join],[dash],[phase])

This sets the line drawing style.
width, is the thickness of the line in user units
cap is the type of cap to put on the line, values can be 'butt','round','square' where the difference between 'square' and 'butt' is that 'square' projects a flat end past the end of the line.

join can be 'miter', 'round', 'bevel'
dash is an array which sets the dash pattern, is a series of length values, which are the lengths of the on and off dashes.
for example: (2) represents 2 on, 2 off, 2 on , 2 off ...
(2,1) is 2 on, 1 off, 2 on, 1 off.. etc
phase is a modifier on the dash pattern which is used to shift the point at which the pattern starts.

```
$pdf->setLineStyle(1);
$pdf->setLineStyle(5);
$pdf->setLineStyle(5,'round');
$pdf->setLineStyle(5,'','',array(5));
$pdf->setLineStyle(5,'','',array(10,5));
$pdf->setLineStyle(5,'','',array(20,5,10,5));
$pdf->setLineStyle(5,'round','',array(0,15));
```

Note that the code shown with each of these lines is just the line style command, the drawing of each line also requires a **line** command, which is not shown.

line (x1,y1,x2,y2)

Draw a line from (x1,y1) to (x2,y2), set the line width using setLineStyle.

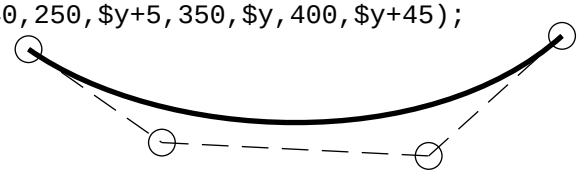
curve (x0,y0,x1,y1,x2,y2,x3,y3)

Draw a Bezier curve. The end points are (x0,y0)->(x3,y3), and the control points are the other two.

Bezier curves are neat, but probably not for the novice. The **ellipse** command uses a series of Bezier curves to make the ellipse.

The distinctive feature of these curves is that the curve is guaranteed to lie within the 4 sided polygon formed from the 4 control points (they are also computationally easy to draw). An example is shown below with the control points marked.

```
$pdf->curve(200,$y+40,250,$y+5,350,$y,400,$y+45);
```



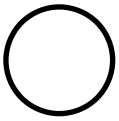
Note that the Bezier curve is a tangent to the line between the control points at either end of the curve.

ellipse (x0,y0,r1,[r2=0],[angle=0],[nSeg=8])

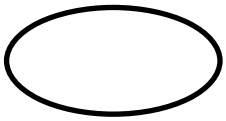
Draw an ellipse, centred ay (x0,y0), with radii (r1,r2), oriented at 'angle' (anti-clockwise), and formed from nSeg bezier curves (the default 8 gives a reasonable approximation to the required shape).

If r2 is left out, or set to zero, then it is assumed that a circle is being drawn.

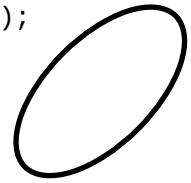
```
$pdf->ellipse(300,$y+25,20);
```



```
$pdf->ellipse(300,$y+25,40,20);
```

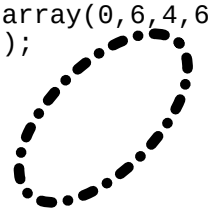


```
$pdf->ellipse(300,$y+25,40,20,45);
```



Of course the previous line style features also apply to these lines

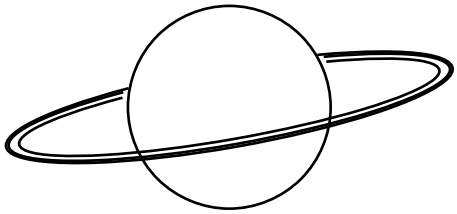
```
$pdf->setLineStyle(4,'round','',array(0,6,4,6));
$pdf->ellipse(300,$y+25,40,20,45);
```



partEllipse (x,y,a1,a2,r1 [,r2] [,angle] [,nSeg])

Draw a part ellipse (or circle), draw an ellipse centered on (x,y) from angle 'a1' to angle 'a2' (in degrees), with radius 'r1' in the x-direction and 'r2' in the y-direction and oriented at angle 'angle'. If 'r2' is not supplied the it defaults to 'r1' and a circle is formed.

```
$pdf->ellipse(300,$y+25,38);
$pdf->partEllipse(300,$y+25,119,421,80,12,10);
$pdf->partEllipse(300,$y+25,117,423,84,14,10);
$pdf->partEllipse(300,$y+25,115,425,85,15,10);
```



polygon (p,np,[f=0])

Draw a polygon, where there are np points, and p is an array containing (x0,y0,x1,y1,x2,y2,...,x(np-1),y(np-1)).
If f=1 then fill the area.

```
$pdata = array(200,10,400,20,300,50,150,40);
$pdf->polygon($pdata,4);
```

```
$pdf->polygon($pdata,4,1);
```

```
$pdf->setColor(0.9,0.9,0.9);
$pdf->polygon($pdata,4,1);
```


rectangle (x1,y1,width,height)

Build a rectangle with no background, only borders

```
$pdf->rectangle(400, $pdf->y, 120,50);
```

filledRectangle (x1,y1,width,height)

Build a rectangle with filled color.

```
$pdf->filledRectangle(400, $pdf->y, 120, 50);
```

newPage ([insert,id,pos]) = id

Starts a new page and returns the id of the page contents, this can be safely ignored, but storing it will allow the insertion of more information back into the page later, through the use of the 'reopenObject' function.

The command is usually used without any of the options to simply add a new page to the

end of the current bunch, but with the options can be used to insert a page within the existing pages. The 'insert' value can be set to 0 or 1 (use 1 to insert). 'id' should be set to a value which was returned by a previous newPage command (this is actually the object id of the contents of a page). 'pos' will determine whether the page is inserted before or after the specified page, it should be set to 'before' or 'after'.

getFirstPageId () = id

A related command is this which returns the id of the first page, this page is created during the class instantiation and so does not have its id returned to the user, this is the only way to fetch it, but it can be done at any point.

stream ([array options])

Used for output, this will set the required headers and output the pdf code.

The options array can be used to set a number of things about the output:

'Content-Disposition'=>'filename' sets the filename, though not too sure how well this will work as in my trial the browser seems to use the filename of the php file with .pdf on the end.
'Accept-Ranges'=>1 or 0 - if this is not set to 1, then this header is not included, off by default this header seems to have caused some problems despite the fact that it is supposed to solve them, so I am leaving it off by default.
'compress'=> 1 or 0 - apply content stream compression, this is on (1) by default.

getFontHeight (size) = height

Returns the height of the current font, in the given size. This is the distance from the bottom of the descender to the top of the Capitals.

getFontDescender (size) = height

Returns a number which is the distance that the descender goes beneath the Baseline, for a normal character set this is a negative number.

getTextWidth (size,text) = width

Returns the width of the given text string at the given size.

saveState ()

Save the graphic state.

restoreState ()

Restore a saved graphics state.

openObject () = id

Start an independent object. This will return an object handle, and all further writes to a page will actually go into this object, until a closeObject call is made.

reopenObject (id)

Makes the point of current content insertion the numbered object, this 'id' must have been returned from a call to 'openObject' or 'newPage' for it to be a valid object to insert content into. Do not forget to call 'closeObject' to close off input to this object and return it to where it was beforehand (most likely the current page).
This will allow the user to add information to previous pages, as long as they have stored the id of the pages.

closeObject ()

Close the currently open object. Further writes will now go to the current page.

addObject (id,[options='add'])

Add the object specified by id to the current page (default). If a string is supplied in options, then the following may be specified:
'add' - add to the current page only.
'all' - add to every page from the current one on.
'odd' - add to all odd numbered pages from now on.
'even' - add to all even numbered pages from now on.
'next' - add to just the next page.
'nextodd' - add to all odd numbered pages from the next one.
'nexteven' - add to all even numbered pages from the next one.

stopObject (id)

If the object (id) has been appearing on pages up to now, then stop it, this page will be the last one that could contain it.

addInfo (label,value)

Add document information, the valid values for label are:
Title, Author, Subject, Keywords, Creator, Producer, CreationDate, ModDate, Trapped

modified in version 003 so that 'label' can also be an array of key->value pairs, in which case 'value' should not be set.

setPreferences (label,value)

Set some document preferences, the valid values for label are:
HideToolbar, HideMenuBar, HideWindoUI, FitWindow, CenterWindow, NonFullScreenPageMode, Direction

modified in version 003 so that 'label' can also be an array of key->value pairs, in which case

'value' should not be set.

addImage (img,x,y,w,[h],[quality=75],[angle=0])

Add an image to the document, this feature needs some development. But as it stands, img must be a handle to a GD graphics object, and one or both of w or h must be specified, if only one of them is specified, then the other is calculated by keeping the ratio of the height and width of the image constant.
The image will be placed with its lower left corner at (x,y), and w and h refer to page units, not pixels.

addJpegFromFile (imgFileName,x,y,w,[h],[angle=0])

Add a JPEG image to the document, this function does not require the GD functionality, so should be usable to more people, interestingly it also seems to
be more reliable and better quality. The syntax of the command is similar to the above 'addImage' function, though 'img' is a string containing the file name of the jpeg image.

x,y are the position of the lower left corner of the image, and w,h are the width and height. Note that the resolution of the image in the document is defined only by the resolution of the image that you insert, and the size that you make it on the page. If you have an image which is 500 pixels across and then you place it on the page so that it is 72 units across then this image will be about 500 dpi (as 72 units is about 72 points which is 1 inch).

addPngFromFile (imgFileName,x,y,w,[h],[angle=0])

Similar to *addJepgFromFile*, but for PNG images.

output ([debug=0]) = a

As an alternative to streaming the output directly to the browser, this function simply returns the pdf code. No headers are set, so if you wish to stream at a later time, then you will have to manually set them. This is ideal for saving the code to make a pdf file, or showing the code on the screen for debug purposes.

If the 'debug' parameter is set to 1, then the only effect at the moment is that the compression option is not used for the content streams, so that they can be viewed clearly.

openHere (style[,a][,b][,c])

Make the document open at a specific page when it starts. The page will be the one which is the current page when the function is called.

The style option will define how it will look when open, valid values are:
(where the extra parameters will be used to supply any values specified for the style chosen)

'XYZ' left, top, zoom *open at a particular coordinate on the page, with a given zoom factor*
'Fit' *fit the page to the view*
'FitH' top *fit horizontally, and at the vertical position given*

'FitV' left *fit vertically, and at the horizontal position given*
'FitB' *fit the bounding box of the page into the viewer*
'FitBH' top *fit the width of the bounding box to the viewer and set at the vertical position given*
'FitBV' left *fit bounding box vertically and set given horizontal position*

selectFont (fontName [,encoding = ''] [, set = 1] [,subsetFont = false])

This selects a font to be used from this point in the document. Errors can occur if some of the other functions are used if a font has not been selected, so it is wise to do this early on.

In all versions prior to 0.12 it was required to "convert" ttf fonts into either *.AFM or *.UFM for unicode font files.
Since version >= 0.12.0 R&OS pdf class natively reads the information from the TTF font by using TTF.php
It also supports font supbsetting (>= 0.11.8) by using TTFsubset.php. Thanks to Thanos Efraimidis (4real.gr).

The experimental version 0.12 currently does not support Type 1 fonts

The encoding directive is still experimental and no guarantee its working at all in version >= 0.12. But theoretical it allows the user to re-map character numbers from the 0->255 range to any named character in the set (as most of the sets have more than 256 characters). It should be an array of <number> => <name> pairs in an associative array. This is important to ensure that the right width for the character is used within the presentation characters, it has been noticed that sometimes, although the right character appears on the page, the incorrect width has been caluclated, using this function to explicitly set that number to the named character should fix the problem.
Note that the encoding directive will be effective **only the first time** that a font is selected, and it should not be used on symbolic fonts (such as Symbol or ZapfDingbats).

```
// use a Times-Roman font with MacExpertEncoding
$pdf->selectFont('Times-Roman','MacExpertEncoding');
// this next line should be equivalent
$pdf->selectFont('Times-Roman',array('encoding'=>'MacExpertEncoding'));

// setup the helvetica font for use with german characters
$diff=array(196=>'Adieresis',228=>'adieresis',
            214=>'Odieresis',246=>'odieresis',
            220=>'Udieresis',252=>'udieresis',
            223=>'germandbls');
$pdf->selectFont('Helvetica'
       ,array('encoding'=>'WinAnsiEncoding'
            ,'differences'=>$diff));
```

setFontFamily (family,options)

This function defines the relationship between the various fonts that are in the system, so that when a base font is selected the program then knows which to use when it is asked for the **bold** version or the *italic* version (not forgetting of course ***bold-italic***).

It maintains a set of arrays which give the alternatives for the base fonts. The defaults that are in the system to start with are:

'Helvetica'
'b'=>'Helvetica-Bold'
'i'=>'Helvetica-Oblique'
'bi'=>'Helvetica-BoldOblique'
'ib'=>'Helvetica-BoldOblique'

'Courier'
'b'=>'Courier-Bold'
'i'=>'Courier-Oblique'
'bi'=>'Courier-BoldOblique'
'ib'=>'Courier-BoldOblique'

'Times-Roman'
'b'=>'Times-Bold'
'i'=>'Times-Italic'
'bi'=>'Times-BoldItalic'
'ib'=>'Times-BoldItalic'

(where 'b' indicate bold, and 'i' indicates italic)

Which means that (for example) if you have selected the 'Courier' font, and then you use the italic markers then the system will change to the 'Courier-Oblique' font.

Note that at the moment it is not possible to have any more fonts, so there is little use in calling this function (except to change these settings, discussed below), but this is paving the way for when soon we will be able to add other fonts (both .afm, and .ttf), which is also why the suffix must be specified in the font name.

Note also that it is possible to have a different font when some text is bolded, then italicized, as compared to when it is italicized, then bolded ('bi' vs 'ib'), though they have all been set to be the same by default.

If you bold a font twice, then under the current system it would look for a 'bb' entry in the font family, and since there are none in the default families the font will revert to the base font. As an example here is the code that you would use to define a new font family for the Courier font which (for some reason) changed to Times-Roman when a double bold is used:

```
$tmp = array(
          'b'=>'Courier-Bold'
         ,'i'=>'Courier-Oblique'
         ,'bi'=>'Courier-BoldOblique'
         ,'ib'=>'Courier-BoldOblique'
         ,'bb'=>'Times-Roman'
       );
$pdf->setFontFamily('Courier',$tmp);
```

setEncryption ([userPass=''],[ownerPass=''],[pc=array], [mode=1])

Calling this function sets up the document to be encrypted, this is the only way to mark the document so that they user cannot use cut and paste, or printing.

Since version 0.11.2 a new parameter "$mode" has been added to this function to allow RC4 128bit encryption (PDF 1.4 required)

Using the call without options, defaults to preventing the user from cut & paste or printing. There are no passwords require to open the document.

```
$pdf->setEncryption();
```

Setting either off the passwords will mean that the user will have to enter a password to open the document. If the owner password is entered when the document is opened then the user will be able to print etc. If the two passwords are set to be the same (or the owner password is left blank) then there is noo owner password, and the document cannot be opened in the accesible mode.

The pc array can be used to **allow** specific actions. The following example, sets an owner password, a user password, and allows printing and cut & paste.

```
$pdf->setEncryption('trees','frogs',array('copy','print'));
```

addLink (url,x0,y0,x1,y1)

Creates a clickable rectangular area within the document, which takes the user to the URL when clicked. The coordinates specify the area.

See the information in the *inline codes* chapter of the ezPdf section to see and easy way of adding links to your code.

```
$pdf->addLink("https://github.com/rospdf/pdf-php/",50,100,500,120);
$pdf->rectangle(50,100,450,20);
```

addInternalLink (label,x0,y0,x1,y1)

Creates an internal link in the document, 'label' is the name of an target point, and the other settings are the coordinates of the enclosing rectangle of the clickable area.

Note that the destination markers are created using the *addDestination* function described below.

See the information in the *inline codes* chapter of the ezPdf section to see and easy way of adding links to your code.

addDestination (label,style[,a][,b][,c])

This marks a point in the document as a potential destination for an internal link, the 'label' can be any string, though should be unique for the document, else confusion may ensue. The remainer of the options are identical to the *openHere* function.

transaction (action)

One of the major problems with this class has been the inability to format things so that they fit nicely, as you cannot know how large something is until you have it on the page, this function attempts to produce a solution this.

Transaction support (terminology borrowed from databases) allows you to mark a point in the development of your document, and if you do not like how things are going from there you can abort and return to that place. This will allow tentatively trying a few different options for things, then settling with the one that looks like what you like. The most obvious application immediatly is support for stopping table cells being split over pages. If you start a row and on completion discover that you are on a new page, then just abort the transaction, going back to the position before you started making that row, and then make a new page there.

'action' can be set to one of 'start','commit','rewind','abort', which are fairly obvious in their usage.

'start' - mark a checkpoint, a position to return to upon 'abort'
'commit' - you are happy with this one, it releases the most recent 'start' though does not affect the document.
'rewind' - you are not happy, but are planning to start again from the last checkpoint, this is the same as an 'abort' and a 'start', but is much more efficient.
'abort' - you are not happy with you the document is looking, this will return you to the state at last 'start' point.

Notes:
- 'start' commands can be nested, and 'abort' or 'commit' will always affect the most recent start.
- though often it seems that you do not need to 'commit' if you are happy with your changes, it is good practice, the 'start' command is effectively making an internal copy of the document, if you are working on a large document you may run out of space. The 'commit' command frees up that space once more.
- these commands, though they are part of the base class, save all the setting of the object in which they are contained, this means most often that they will work fine with an ezPdf object, and will also work with user class extensions.

Misc

Callback functions

Code has been included within class.pdf.php which allows the user to place a marker within the text, when the interpreter places that piece of text on the page, then the named function will be called, passing an array which has the details about the position.

The marker is either of the forms:

```
<c:func:paramater>aaa bbb ccc</c:func>
<C:func:paramater>
```

The first form (with the small 'c') is for when the situation requires both an opening and a closing tag, and the second form for when only one specific point needs to be marked.

When this piece of text is placed, the class function 'func' will be called, and it must have been defined with a single paramater. This paramater will be an array, which will have members:

```
x => x-position
y => y-position
status => <see below>
f => function name
height => font height
decender => font decender
```

Where the status can have one of the values 'start','end','sol','eol'. The start and end values are obvious and indicate that this is either the start or the end of the marked range. The other two indicate start-of-line and end-of-line, which are called if the start or end of a line is reached between matching start and end markers.

Note that the function needs to be a function of the pdf class, so if you wish to use a custom function then you need to extend the class and add your own functions. This is done within the script that makes this document, to collect the information which makes the table of contents, and to add the dots on each line within the table of contents display.

NOTE For various technical reasons if there are any spaces within the marker, then these will affect the justification calculation, it is best to avoid having them. This is why the title names in the example below were urlencoded when placed in the paramater section.

NOTE These functions should be used with care, it is possible to set up an infinite loop. If the function called by the two-part form has an addText command within it, then if it is not set up very carefully then an infinite loop is formed as the function is re-called at the start and end of the addText command... and so on... and so on... and so on...

As an example, here is the code which extends the class for this document

```
include 'class.ezpdf.php';

class Creport extends Cezpdf {

// make a location to store the information that will be collected during
document formation
var $reportContents = array();

// it is neccesary to manually call the constructor of the extended class
function Creport($p,$o){
  $this->Cezpdf($p,$o);
}

/*
The function 'rf' records in an array the page number level number and
heading for each title as the document is constructed. This information is
used at the end to construct the table of contents. Each heading has had a
marker put next to it of the form:
<C:rf:1top%20heading>, which would be for a level 1 heading called 'top
heading'.
After the bulk of the document has been constructed, the array $pdf
->reportContents is selected, and the table of contents made.
 */
function rf($info){
  $tmp = $info['p'];
  $lvl = $tmp[0];
```

```
  $lbl = rawurldecode(substr($tmp,1));
  $num=$this->ezWhatPageNumber($this->ezGetCurrentPageNumber());
  $this->reportContents[] = array($lbl,$num,$lvl );
}

/*
The dots function is called by a marker of the form:
<C:dots:213>
Which would represent a heading being show for which the original document
was on page 13, and the heading was level 2. The marker is placed at the
end of the text being displayed within the table of contents, upon
activation it draws a dotted line across from that point to the right hand
side of the page, and puts the given label there (to be more accurate it
draws the line from the right, back to the left, so that all the dots line
up down the page).
*/
function dots($info){
  // draw a dotted line over to the right and put on a page number
  $tmp = $info['p'];
  $lvl = $tmp[0];
  $lbl = substr($tmp,1);
  $xpos = 520;

  switch($lvl){
    case '1':
      $size=16;
      $thick=1;
      break;
    case '2':
      $size=12;
      $thick=0.5;
      break;
  }
  $this->saveState();
  $this->setLineStyle($thick,'round','',array(0,10));
  $this->line($xpos,$info['y'],$info['x']+5,$info['y']);
  $this->restoreState();
  $this->addText($xpos+5,$info['y'],$size,$lbl);
}
}
```

Both of these functions use only the single point call to a callback function, for an example of the more complicated two-part from, refer to the function 'alink' within class.ezpdf.php which uses callback functions to implement the url links on marked document text.

Units

The units used for positioning within this class are the default pdf user units, in which each unit is roughly equivalent to 1/72 of an inch.

and some additional user contributed notes are (thanks Andrew):

1/72" is 0.3528mm or 1 point
1 point was historically 0.0138 inches, a little under 1/72"
10mm is 28.35 points
A4 is 210 x 297 mm or 595.28 x 841.89 points