Modern PHP Graphics with Cairo

cairo

Michael Maclean

FrOSCon 2011

Who am I?

- PHP developer for a number of years
- Working on some PECL extensions
- Mainly graphics extensions

What is Cairo?

It's a vector graphics library

Good for creating graphics, not so good for manipulating photos

Written by the FreeDesktop.org project, dual-licenced as LGPL v2.1 or MPL v1.1

Vector vs. Raster

- Raster graphics are bitmaps – a big matrix of pixels - this is what photos are generally stored as
  - JPEG, PNG, GIF etc are all raster graphic formats
- Vector graphics are descriptions of the lines, and colours that make up an image
  - SVG, WMF, PostScript and PDF are vector formats

Who uses it?

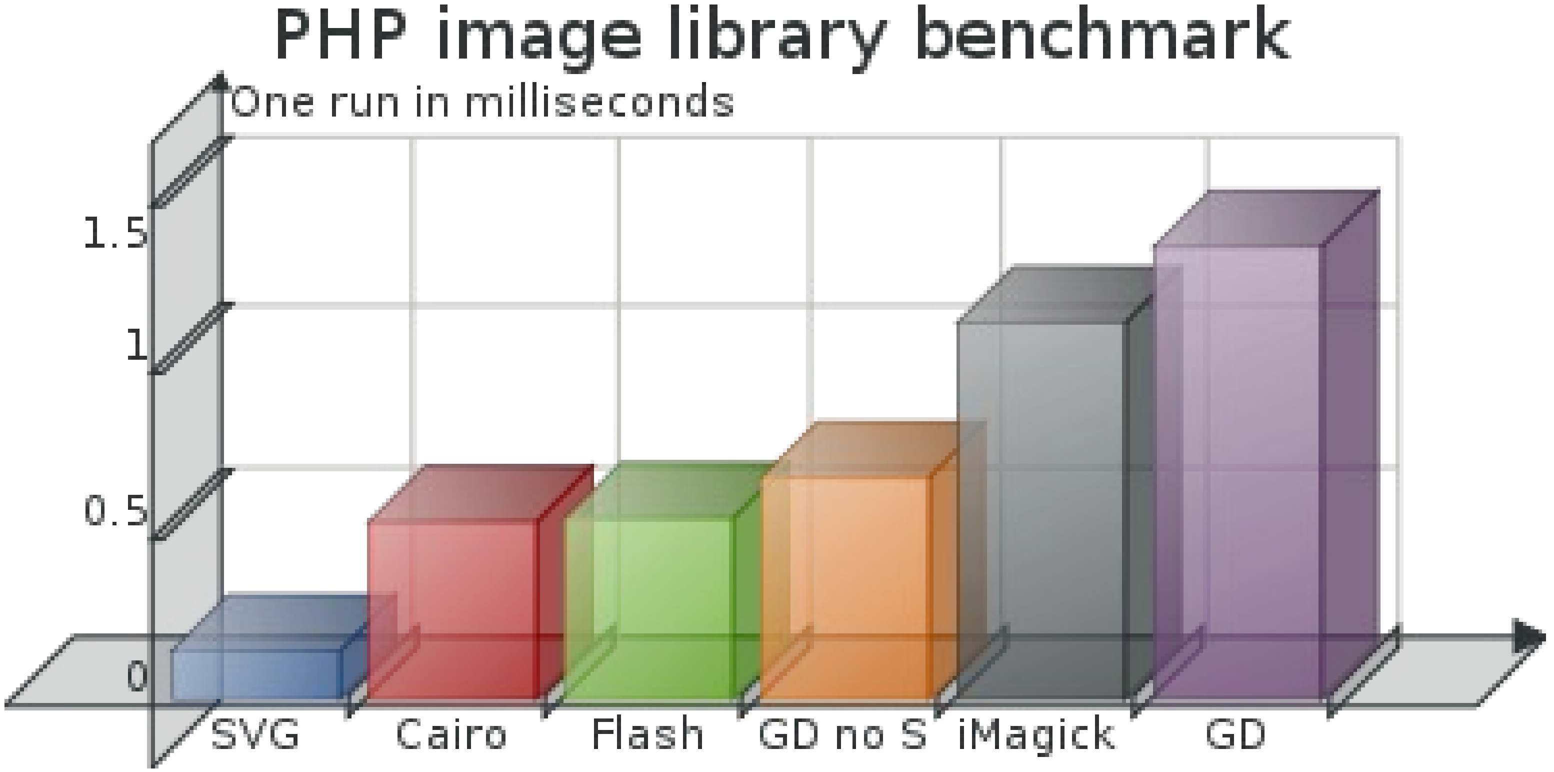
®

GNOME

Why do you want to use it?

There are a number of reasons why you should consider it…

It's fast.



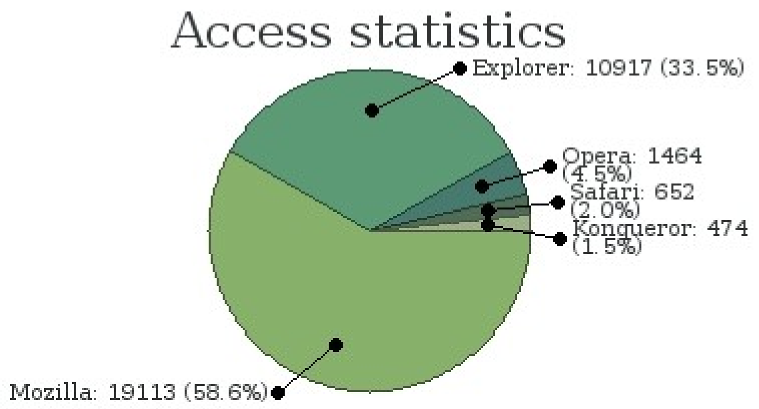
Graphic created by Kore Nordmann

It gives nice output

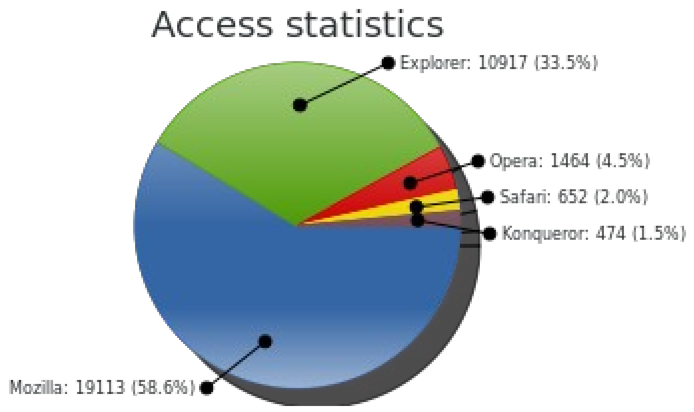
- GD, while it's useful, can't do antialiasing
- It can't do gradients
- It only does raster graphics
- Cairo can and does do all of the above

Output example

GD



Cairo



Graphics generated by ezcGraph

It's Free Software

- If you're using a recent Linux distro, you might well have the libraries installed already
- If not, you can download it for Windows
- (I'm not sure about MacOS X...)

How do you use it with PHP?

There have been several Cairo extensions developed at various times for PHP

PECL/Cairo aims to be “definitive”

It provides both object-oriented and procedural APIs

(I'm only going to demo the object-oriented one)

Installation

- UNIX/Linux – use PECL
- Windows – you can download it from http://perisama.net/cairo/
- Intrepid folk – grab it from PECL SVN and compile it yourself
  - If you do this, please run the tests!

Using Cairo

First, I need to explain a few basics

Surfaces

These are what you draw on

They represent an image of a certain size

You can also use them as the “paint” when drawing on other surfaces

Surfaces

You create them using the constructor:

```
$s = new
CairoImageSurface(CairoFormat::ARGB32,
1000, 1000);
```

This creates a new Image surface, in 32-bit colour, 1000x1000 pixels in size

There are different Surface classes for Images, PDFs, etc.

Surfaces

- The other surface types have different constructors
- PDF, PostScript, and SVG surface constructors take a filename or a file resource
- They don't have colour formats, and output data directly to the file or file resource as you draw
- You can write directly to `php://output` if you like, and send the right header

Contexts

- Contexts are what you use to do the drawing operations.
- They're objects that have methods to draw on surfaces, set colour sources, move around the surface, etc.
- You create a context by passing the constructor an existing CairoSurface object
- `$c = new CairoContext($s);`

Contexts

- Once you have a context, you can draw on your surface.
- The context has methods to set various properties:
    - The colour (referred to as a “source”)
    - The line style, width, end caps, fill style, etc.

Basic context methods

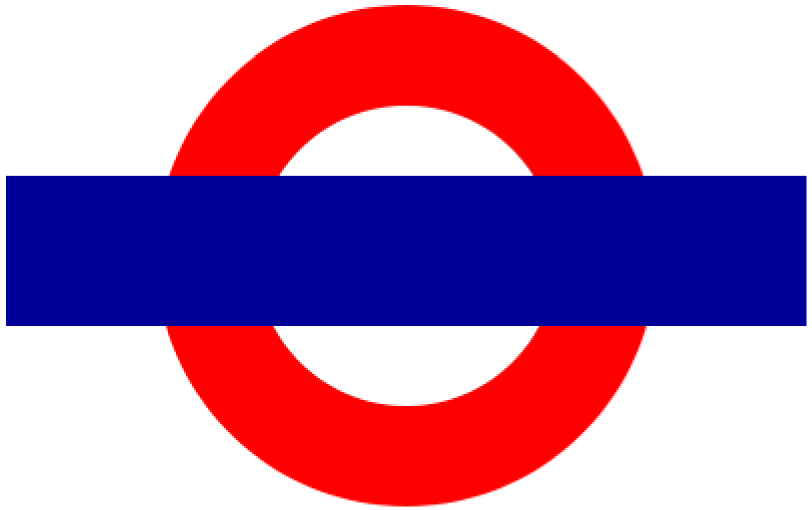
- moveTo() / relMoveTo() - move to a point
- lineTo() - relLineTo() draw a line to a point
- rectangle() - draw a rectangle
- arc() / arcNegative() - draw an arc
- stroke() / strokePreserve() - stroke the current path
- fill() / fillPreserve() - fill the current path

An example

```
$s = new CairoImageSurface(
  CairoFormat::ARGB32, 400, 400);
$c = new CairoContext($s);
$c->fill();
$c->setSourceRGB(1, 0, 0);
$c->setLineWidth(50);
$c->arc(200, 200, 100, 0, 2 * M_PI);
$c->stroke();
$c->setSourceRGB(0, 0, 0.6);
$c->rectangle(0, 160, 400, 75);
$c->fill();

header("Content-type: image/png");
$s->writeToPng("php://output");
```

The result



Where's the text?

- The Cairo library itself supports two APIs for text; the “toy” API and the “real” API
- The toy API is quite sufficient for simple things
- It's also the only one implemented in PECL/Cairo so far
- CairoContext::showText() and CairoContext::textPath() are the main methods

Fonts

- There are a couple of ways of selecting fonts at the moment
- CairoContext::selectFontFace() will attempt to select the font you specify, and lets you choose italic or bold if you want
- The CSS2 names (“sans”, “serif”, “monospace” etc.) are likely to be available anywhere

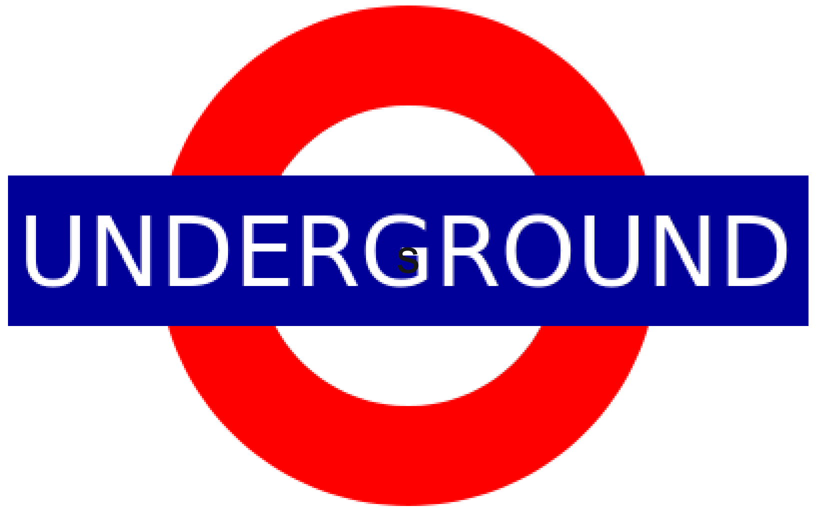
Adding text to the example

After the final fill(), we add:

```
$c->selectFontFace(
  "sans", CairoFontSlant::NORMAL,
  CairoFontWeight::NORMAL);

$c->moveTo(5, 215);
$c->setSourceRGB(1, 1, 1);
$c->setFontSize(48);
$c->showText("UNDERGROUND");
```

The slightly cheesy result



Using local fonts

- There is also a font class that uses FreeType to load any font file that FreeType can read (which is most of them)

```
$s = new
CairoImageSurface(CairoFormat::ARGB32, 200,
  100);
$c = new CairoContext($s);
$f = new CairoFtFontFace("vollkorn.otf");
$c->setFontFace($f);
$c->moveTo(10, 10);
$c->showText("Hello world");
```

Complex text handling

- Cairo doesn't really do text layout, it's designed for graphics
- You have to do the positioning work yourself
- Or...

Pango

Pango is the text layout library generally used with Cairo

Pango

I've written an extension for it too

`http://github.com/mgdm/php-pango`

Pango

Pango is able to handle laying out more complex text than Cairo is

Pango

It's able to set paragraphs, and do line breaking etc

Pango

It has its own HTML-like markup language to handle text attributes like bold, italic, etc

Pango Markup

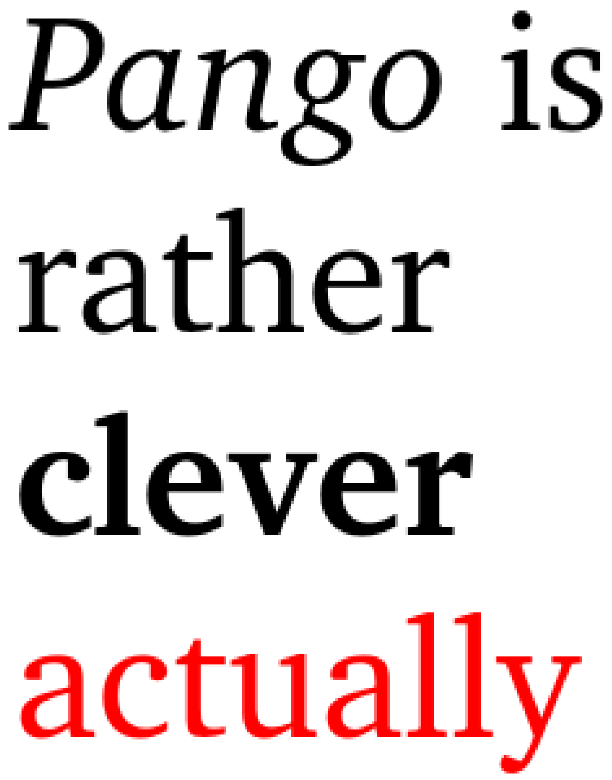
Pango example

```
// $c is a CairoContext
$l = new PangoLayout($c);
```

- `$l->setMarkup("<i>Pango</i> is rather <b>clever</b> actually");`

```
$desc = new PangoFontDescription(
    “DejaVu Sans Mono 24”);
$l->setFontDescription($desc);
$l->showLayout();
```

Pango Example



Pango is
rather
clever
actually

</pango>

Patterns

- Patterns are the “ink” used when drawing
- setSourceRGB() and setSourceRGBA() are shortcuts to set solid colour patterns
- You don't have to use solid colours as sources
- You can create gradients, either linear or radial
- You can use other surfaces, too

Linear gradients

- They are a progression from one colour to another along a line
- You create a CairoLinearGradient object, with a pair of coordinates representing the line
- Then, add stops, specifying the distance along the line and the colour in RGBA
  - Note the line length is normalized to 1.0 when you do that

Linear gradient example

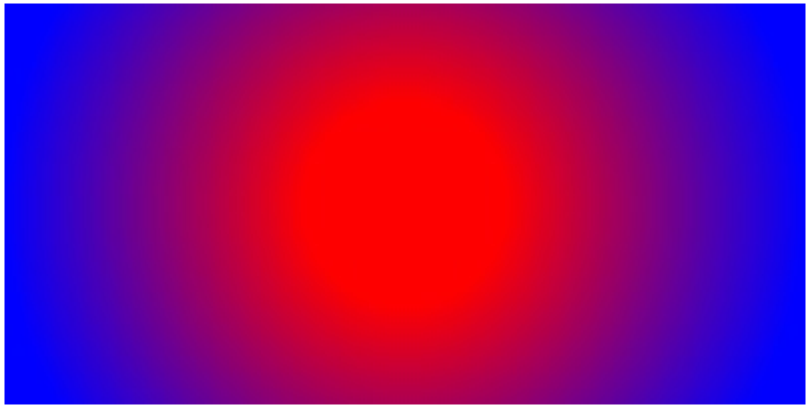
```
$p = new CairoLinearGradient(0, -10, 0, 10);
$p->addColorStopRgba(0,1,0,0,1);
$p->addColorStopRgba(1,0,0,1,0.5);
$c->setSource($pat);
$c->paint();
```

Radial gradients

- These are described as having one circle of one colour inside another
- You pass the XY coordinates of the two circles, and their radii, as arguments to the constructors
- Then you add colour stops, as before

Radial gradient example

```
$p = new CairoRadialGradient(200, 100, 50,
                             200, 100, 200);
$p->addColorStopRGBA(0, 1, 0, 0, 1);
$p->addColorStopRGBA(1, 0, 0, 1, 1);
$c->setSource($p);
$c->paint();
```

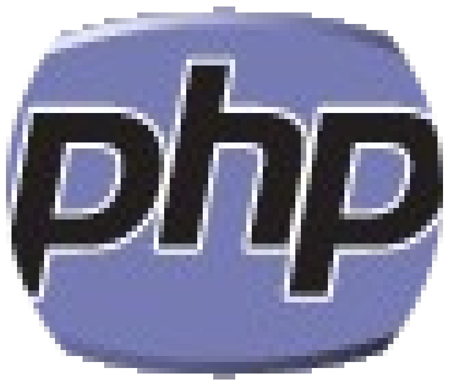


Using other surfaces

- As I mentioned before, you can use other surfaces as sources
- You can also create new Image surfaces by loading PNG files
    - (Support for loading other filetypes is on the way...)

Example

```
$source = CairoImageSurface::createFromPNG(
     dirname(__FILE__) . "/php-logo.png");
$c->setSourceSurface($source);
$c->arc(60, 33, 40, 0, 2 * M_PI);
$c->fill();
```



Other surface types

- The other two surfaces that are probably useful are PDF and SVG
- There are a couple of limitations, unfortunately
    - The Cairo library has no way to create anchors in PDFs or SVGs, so you can't create hyperlinks (yet!)
- Both of these are written straight to a file as you create them, which you can overcome with the PHP streams API

Creating PDFs

```
header("Content-Type: application/pdf");
header("Content-Disposition: attachment; filename=cairo-
pdf.pdf");
$s = new CairoPdfSurface("php://output",
                         210 * 2.83, 297 * 2.83);
$c = new CairoContext($s);

$c->setFontSize(48);
$c->moveTo(10, 100);
$c->showText("Hello world");
$c->showPage();

$s->setSize(297 * 2.83, 210 * 2.83);
$c->moveTo(10, 100);
$c->showText("And this should be page 2.");
$c->showPage();
```

More on contexts

- Contexts have an internal stack, which means you can save a state and restore it later
- This is handy in certain situations, but mainly when using…

Transformations

- Contexts let you draw using one set of coordinates, but have them appear on the surface in another
- This is cool where you have a method to draw a certain item that you might want over and over in different places or orientations

Transformations

- CairoContext::translate() moves the origin (normally top left) to somewhere else
- CairoContext::scale() scales the coordinates so what you draw ends up at a different size on the surface
- CairoContext::rotate() rotates the coordinates to draw at a different angle

save() and restore()

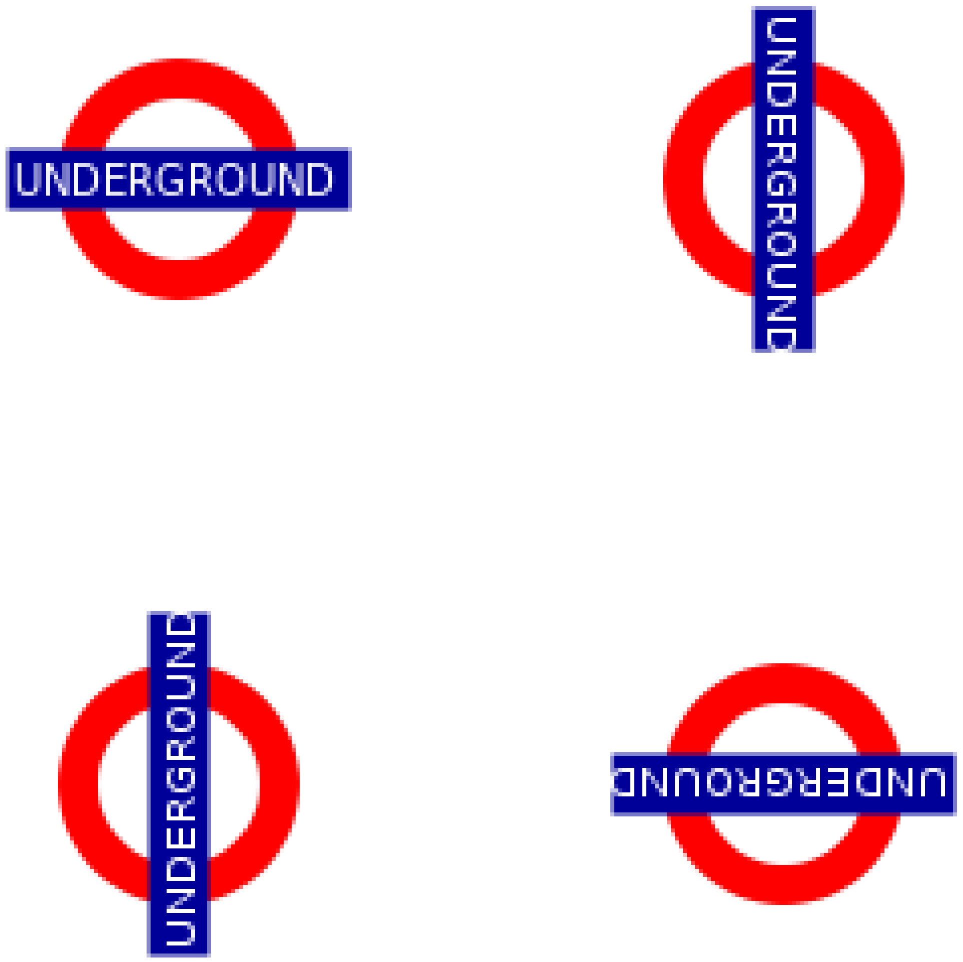
- You can use the stack to save the current transformation, so you can go and do something else and restore later

Transformation example

```
function drawLogo($c) {
    $c->save();
    $c->scale(1.0, 1.0);
    $c->setSourceRGB(1, 0, 0);
    $c->setLineWidth(10);
    $c->arc(0, 0, 25, 0, 2 * M_PI);
    $c->stroke();
    $c->setSourceRGB(0, 0, 0.6);
    $c->rectangle(-42.5, -7.5, 85, 15);
    $c->fill();
    $c->setSourceRGB(1, 1, 1);
    $c->moveTo(-41, 4);
    $c->showText("UNDERGROUND");
    $c->restore();
}

$c->translate(75, 75);
for($i = 0; $i < 4; $i++) {
    drawLogo($c);
    $c->translate(250, 0);
    $c->rotate(M_PI / 2);
}
```

Example output



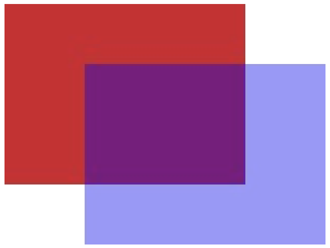
Operators

- Normally, Cairo will draw on top of whatever is on the surface
- You can change this, using the operators, which are specified using CairoContext::setOperator()
- There are a few operators you can choose from

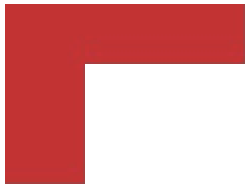
Basic operators

- CairoOperator::OVER – the default – replace destination
- CairoOperator::CLEAR – clear (erase) the destination
- CairoOperator::IN – draw where there is already something on the destination
- CairoOperator::OUT – draw where there **isn't** already something on the destination
- ...etc (check the manual, it's rather dull)

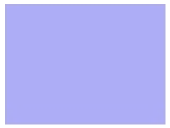
Operator examples



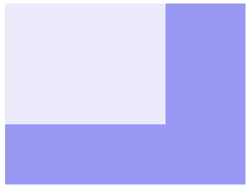
OVER



CLEAR



IN



OUT

New version

We're releasing a new version of the extension shortly

New version

Please check it out and give it a go, running the tests if you can

New version

Feedback is always welcome!

Any questions?

Thank you for listening

- http://kore-nordmann.de/blog/comparision_of_php_image_libraries.html

Slides and demo code will appear online at
http://mgdm.net/talks/

Drop me a line on mgdm@php.net
or Twitter @mgdm
or mgdm on Freenode IRC