Basic Usage

A quick note before we get started. The image used in these examples is not mine, but as far as I know I'm free to use it here. If it's yours and you want credit, or for me to remove it, let me know and I'll take care of it right away.

Here's the original image all these examples will be using:

Showing vs Saving

The library allows you to either show an image after manipulation or save it to the filesystem. We'll cover how to do these specifically later on this page, but let's go over a few important things to know first.

When showing an image, the library sends a header before the image stream. Because of this you CANNOT show an image directly in the middle of your page since output will have already started. It's very much the same as how you can't just toss a `session_start()` in the middle of your page. You can, however, get around this restriction in a relatively easy way, which we'll take a look at later.

Saving images is pretty straight-forward, but it's important to make sure the directory that you're saving the images to is writable by the server. It's also best to use the full, absolute path to the file when saving, rather than relative paths.

Resizing

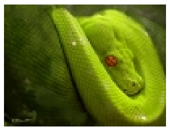
So, let's take a look at some basic resize operations. There are three different ways to resize with the library.

Basic Resizing

To begin, let's assume we want to create a thumbnail from the above image that is no larger than 100×100 pixels. To do that, we'd simply do the following:

```
<?php

require_once 'path/to/ThumbLib.inc.php';

try
{
     $thumb = PhpThumbFactory::create('/path/to/image.jpg');
}
catch (Exception $e)
{
    // handle error here however you'd like
}

$thumb->resize(100, 100);
$thumb->show();

?>
```

Which outputs the following image:



You should note that the actual image dimensions are 100×75 px. This is because the resize function works off a best-fit algorithm. Basically, it will make sure the width is no greater than the provided width and the same for the height.

Resize by Percentage

We can also resize by a certain percentage. Let's take a look at what happens when we resize the original image to 50% of its original size:

```
<?php

require_once 'path/to/ThumbLib.inc.php';

try
{
    $thumb = PhpThumbFactory::create('/path/to/image.jpg');
}
catch (Exception $e)
{
    // handle error here however you'd like
}

$thumb->resizePercent(50);
$thumb->show();

?>
```

Which yields:

Adaptive Resizing

Let's take a look at one more way to resize. Most of the time, you'll want your thumbnails to be uniformly sized. Luckily, there's a function that will do this for us. What it does is resize the image to get as close as possible to the desired dimensions, then crops the image down to the proper size from the center. This is called adaptive resizing. Here's how we use it:

```
<?php

require_once 'path/to/ThumbLib.inc.php';
```

```
try
{
     $thumb = PhpThumbFactory::create('/path/to/image.jpg');
}
catch (Exception $e)
{
     // handle error here however you'd like
}

$thumb->adaptiveResize(175, 175);
$thumb->show();

?>
```

Which gives us the following:

Cropping

The library also contains two handy cropping functions that allow you to crop from the center of the image, or from specific coordinates (useful when paired with JavaScript croppers). Let’s take a look at how these work.

Crop From Center

First, a crop from center:

```
<?php

require_once 'path/to/ThumbLib.inc.php';

try
{
     $thumb = PhpThumbFactory::create('/path/to/image.jpg');
}
catch (Exception $e)
{
     // handle error here however you'd like
}

$thumb->cropFromCenter(200, 100);
$thumb->show();

?>
```

Which will give us the following image:

We can also quickly create square crops by providing only one parameter to the function:

```
<?php
```

```
require_once 'path/to/ThumbLib.inc.php';

try
{
     $thumb = PhpThumbFactory::create('/path/to/image.jpg');
}
catch (Exception $e)
{
     // handle error here however you'd like
}

$thumb->cropFromCenter(200);
$thumb->show();

?>
```

Which will give us the following image:

Vanilla Cropping

You can also perform a regular ol' crop. Simply give the function your starting x & y coordinates and the desired width and height:

```
<?php

require_once 'path/to/ThumbLib.inc.php';

try
{
     $thumb = PhpThumbFactory::create('/path/to/image.jpg');
}
catch (Exception $e)
{
     // handle error here however you'd like
}

$thumb->crop(100, 100, 300, 200);
$thumb->show();

?>
```

The result:

An important thing to note about all crops. If your width, height, or coordinates are outside the image dimensions, the class will try its best to adjust them to fit the actual dimensions of the image.

Rotating Images

You can also rotate images. Some versions of GD don't have image rotation enabled (this function is only available if PHP is compiled with the bundled version of the GD library). If this is the case, the rotate functions will throw an exception letting you know this is the case.

90 Degree Rotations

The easiest way to rotate an image is either 90 degress clockwise or counter-clockwise.

```
<?php

require_once 'path/to/ThumbLib.inc.php';

try
{
     $thumb = PhpThumbFactory::create('/path/to/image.jpg');
}
catch (Exception $e)
{
     // handle error here however you'd like
}

$thumb->rotateImage('CW');
// or:
// $thumb->rotateImage('CCW');
$thumb->show();

?>
```

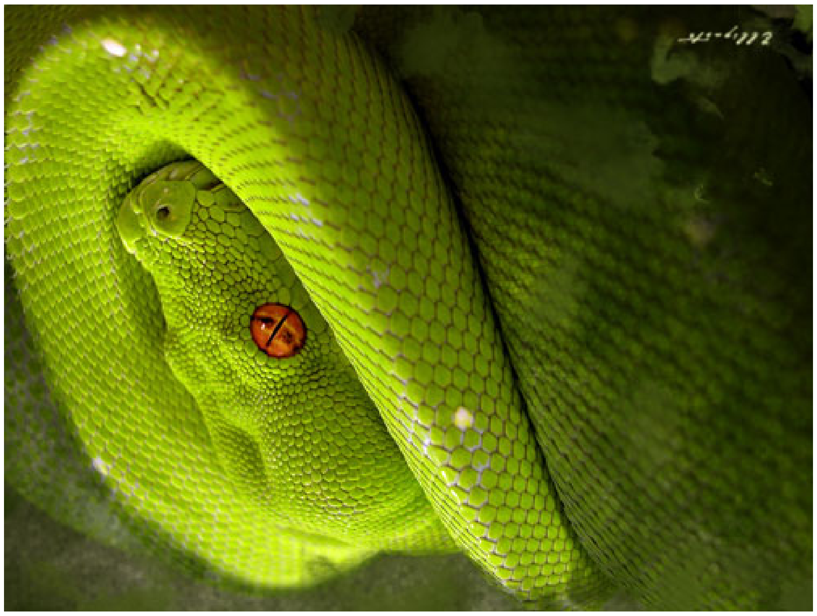
And, you guessed it, we get this:

Arbitrary Rotations

You can also rotate by an arbitrary amount of degrees. Here’s a quick sample:

```
<?php

require_once 'path/to/ThumbLib.inc.php';

try
{
     $thumb = PhpThumbFactory::create('/path/to/image.jpg');
}
catch (Exception $e)
{
     // handle error here however you'd like
}

$thumb->rotateImageNDegrees(180);
$thumb->show();

?>
```

Which gives us:



Saving Images

Saving images is pretty easy. You need only pass the full path to where you wish to save the image (including the file name) the the save function:

```
<?php

require_once 'path/to/ThumbLib.inc.php';

try
{
     $thumb = PhpThumbFactory::create('/path/to/image.jpg');
}
catch (Exception $e)
{
     // handle error here however you'd like
}

// do your manipulations
```

```
$thumb->save('/path/to/new_image.jpg');

?>
```

Easy, right?

You can also change the file format of the original image when you save. Simply pass the desired format as a second parameter to the save function. For example:

```
<?php

require_once 'path/to/ThumbLib.inc.php';

try
{
     $thumb = PhpThumbFactory::create('/path/to/image.jpg');
}
catch (Exception $e)
{
     // handle error here however you'd like
}

// do your manipulations

$thumb->save('/path/to/new_image.png', 'png');

?>
```

Valid formats are:

- png
- jpg
- gif

Note: If you go from an alpha to non-alpha image (i.e. PNG => JPG), you may see weird black matting. This is a known issue and will be addressed in future releases

---

Showing Images

Showing images, at this point, should appear to be pretty easy. But remember, there's a catch. Since showing an image involves sending a header, you can't exactly just drop a `$thumb->show()` in the middle of your pages. You can, however, write a PHP file that shows images for you. Let's imagine you want to show a list of dynamically created thumbnails on a page. Well, we can write a script that does this on-the-fly, and then incorporate that into the page. Let's pretend this script is named "show_image.php" and it contains the following:

```
<?php

require_once 'path/to/ThumbLib.inc.php';

$fileName = (isset($_GET['file'])) ? urldecode($_GET['file']) : null;

if ($fileName h3. null || !file_exists($fileName))
{
     // handle missing images however you want... perhaps show a default image??  Up to you...
}

try
{
     $thumb = PhpThumbFactory::create($fileName);
}
catch (Exception $e)
{
     // handle error here however you'd like
}
```

```
$thumb->adaptiveResize(80, 80);

$thumb->show();

?>
```

Now, all you need to do in the page you wish to display these images in is create image tags that point to the show_image.php file. For example:

```
<img src="show_image.php?file=<?php echo urlencode('/path/to/image.jpg'); ?>" />
```

I personally don't recommend doing this for large lists of images, as your pages will produce significant load on your server. However, this works great for showing a few images (or perhaps a one-off preview of something).

Chaining Functions

One last bit of functionality that we should take a look at is the ability to chain function calls with the Library. Say you want to resize your images down to 150×200 pixels, then crop this image from the center to a 100×100 pixel square. Well, without chaining, your code would look like:

```
<?php

require_once 'path/to/ThumbLib.inc.php';

try
{
     $thumb = PhpThumbFactory::create('/path/to/image.jpg');
}
catch (Exception $e)
{
     // handle error here however you'd like
}

$thumb->resize(150, 200);
$thumb->cropFromCenter(100);
$thumb->show();

?>
```

But, we can clean that up a bit and chain these functions together:

```
<?php

require_once 'path/to/ThumbLib.inc.php';

try
{
     $thumb = PhpThumbFactory::create('/path/to/image.jpg');
}
catch (Exception $e)
{
     // handle error here however you'd like
}

$thumb->resize(150, 200)->cropFromCenter(100)->show();

?>
```

Neat, huh?