

# PHP Solutions: Dynamic Web Design Made Easy

David Powers



# PHP Solutions: Dynamic Web Design Made Easy

Copyright © 2006 by David Powers

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN-13 (pbk): 978-1-59059-731-6

ISBN-10 (pbk): 1-59059-731-1

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail [orders-ny@springer-sbm.com](mailto:orders-ny@springer-sbm.com), or visit [www.springeronline.com](http://www.springeronline.com).

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail [info@apress.com](mailto:info@apress.com), or visit [www.apress.com](http://www.apress.com).

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is freely available to readers at [www.friendsofed.com](http://www.friendsofed.com) in the Downloads section.

## Credits

**Lead Editor**      **Senior Production Editor**  
Chris Mills      Laura Cheu

**Technical Reviewer**      **Composer**  
Samuel Wright      Molly Sharp

**Editorial Board**      **Artist**  
Steve Anglin, Ewan Buckingham, Gary Cornell, Jason  
Gilmore, Jonathan Gennick, Jonathan Hassell, James  
Huddleston, Chris Mills, Matthew Moodie, Dominic  
Shakeshaft, Jim Sumser, Keir Thomas, Matt Wade      April Milne

**Proofreader**  
Liz Welch

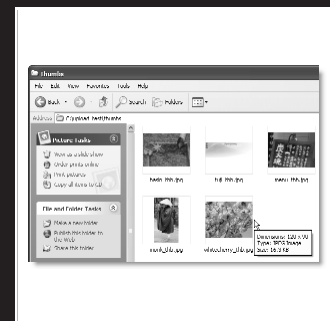
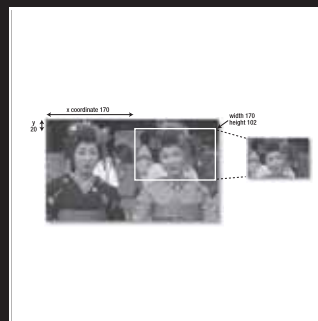
**Senior Project Manager**      **Indexer**  
Kylie Johnston      John Collin

**Copy Edit Manager**      **Interior and Cover Designer**  
Nicole Flores      Kurt Krames

**Copy Editors**      **Manufacturing Director**  
Nicole Flores, Ami Knox      Tom Debolski

**Assistant Production Director**      **Cover Photography**  
Kari Brooks-Copony      David Powers

## 8 GENERATING THUMBNAIL IMAGES



What this chapter covers:

- Scaling an image
- Saving a rescaled image
- Automatically resizing and renaming uploaded images

PHP has an extensive range of functions designed to work with images. You've already met one of them, `getimagesize()`, in Chapter 4. As well as providing useful information about an image's dimensions, PHP can manipulate images by resizing or rotating them. It can also add text dynamically without affecting the original; it can even create images on the fly.

To give you just a taste of PHP image manipulation, I'm going to show you how to generate a smaller copy of an uploaded image. Most of the time, you'll want to use a dedicated graphics program, such as Photoshop or Fireworks, to generate thumbnail images because it gives you much better quality control. However, automatic thumbnail generation with PHP can be very useful if you want to allow registered users to upload images, but make sure that they conform to a maximum size. You can save just the resized copy, or the copy along with the original.

## Checking your server's capabilities

Working with images in PHP relies on the GD extension. Originally GD stood for GIF Draw, but support for GIF files was dropped in favor of JPEG and PNG because of a dispute over a patent. However, the name GD stuck, even though it no longer stands for anything. The problematic patent has now expired and GIF is once again supported, but you need to make sure GD has been enabled on your server and check which features are available.

As in previous chapters, load a page containing `<?php phpinfo(); ?>` to check the server's configuration. Scroll down until you reach the section shown in the following screenshot (it should be about halfway down the page).

gd	
GD Support	enabled
GD Version	bundled (2.0.28 compatible)
FreeType Support	enabled
FreeType Linkage	with freetype
FreeType Version	2.1.9
T1Lib Support	enabled
GIF Read Support	enabled
GIF Create Support	enabled
JPG Support	enabled
PNG Support	enabled
WBMP Support	enabled
XBM Support	enabled

If you can't find this section, it means that the GD extension isn't enabled, so you won't be able to use any of the scripts in this chapter. Your next move depends on your situation.

- On a hosting company's shared server, there's nothing you can do about it, apart from complain or move to a different host.
- If you're checking your local testing environment on a Windows computer, open `php.ini` and locate the following line in the list of Windows extensions:  

```
;extension=php_gd2.dll
```

Remove the semicolon at the start of the line, save `php.ini`, and restart Apache or IIS. If you still can't see that GD support has been enabled, refer back to Chapter 2. Make sure that the correct version of `php.ini` is being read, `extension_dir` is pointing to the correct location, and your Windows path setting includes your PHP folder.

- On a Mac, GD is enabled by default in the package created by Marc Liyanage that I recommended in Chapter 2.

Assuming that GD support is enabled on your server, check the version and the settings for GIF Read Support, GIF Create Support, JPG Support, and PNG Support. GD Version needs to be a minimum of 2. All versions should support JPEG and PNG files, but you need 2.0.28 or later for full GIF support. If the version number is lower than 2.0.28, you will probably be able to read GIF files, but not create them. The scripts in this chapter have been designed to respond appropriately to different levels of support.

*Strictly for abbreviation/acronym freaks: GIF stands for Graphics Interchange Format, JPEG is the standard created by the Joint Photographic Experts Group, and PNG is short for Portable Network Graphics. Although JPEG is the correct name for the standard, the "E" is frequently dropped, particularly when used as a filename extension.*

## Manipulating images dynamically

The GD extension allows you to generate images entirely from scratch or work with existing images. Either way, the underlying process always follows four basic steps:

1. Create a resource for the image in the server's memory while it's being processed.
2. Process the image.
3. Display and/or save the image.
4. Remove the image resource from the server's memory.

This process means that you are always working on an image in memory only and not on the original. Unless you save the image to disk before the script terminates, any changes are discarded. Working with images requires a lot of memory, so it's vital to destroy the image resource as soon as it's no longer needed. If a script runs very slowly or crashes, it probably indicates that the original image is too large.

## Making a smaller copy of an image

The aim of this chapter is to show you how to resize images automatically on upload. This involves adapting the file upload form from Chapter 6. However, to make it easier to understand how to work with PHP's image manipulation functions, I propose to start by using images already on the server, and merge the resizing script with the upload code only at the final stage.

### Getting ready

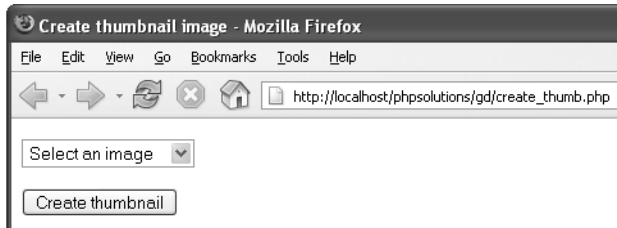
The starting point is the following simple form, which uses the `buildFileList()` function from the last chapter to create a drop-down menu of the photos in the `images` folder. You can find the code in `create_thumb01.php` in the download files for this chapter. Copy it to a new folder called `gd` in the `phpsolutions` site root, and rename it `create_thumb.php`.

```
<?php
// execute script only if the form has been submitted
if (array_key_exists('create', $_POST)) {
    // image resizing script goes here
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<title>Create thumbnail image</title>
</head>

<body>
<form id="form1" name="form1" method="post" action="">
  <p>
    <select name="pix" id="pix">
      <option value="">Select an image</option>
    </select>
  </p>
  <p>
    <input name="create" id="create" type="submit"
value="Create thumbnail" />
  </p>
</form>
</body>
</html>
```

The page requires `buildFileList5.php` (or `buildFileList4.php`, if you're using PHP 4), which should already be in your `includes` folder from the previous chapter. If you don't have a copy, get it from the download files for Chapter 7. Use `buildListFile5()` for PHP 5 and `buildListFile4()` for PHP 4.

When loaded into a browser, the form looks like the screenshot to the right, and the drop-down menu should display the names of the photos in the `images` folder, as shown in Figure 7-1 in the previous chapter.



Inside the `upload_test` folder that you created in Chapter 6, create a new folder called `thumbs`, and make sure it has the necessary permissions for PHP to write to it. Refer back to “Establishing an upload directory” in Chapter 6 if you need to refresh your memory.

## Building the script

Once you have created the `thumbs` folder and checked that the drop-down menu in `create_thumb.php` is displaying a list of images, you're ready to start.

### PHP Solution 8-1: Calculating the scaling ratio

1. If you have been reading the chapters in order, you'll know by now that the conditional statement above the `DOCTYPE` declaration checks whether the `name` attribute of the submit button is in the `$_POST` array. Since the submit button is called `create`, the script inside the conditional statement runs only if the form has been submitted. Replace the placeholder comment with the following code:

```
if (array_key_exists('create', $_POST)) {
    // define constants
    define('SOURCE_DIR', 'C:/htdocs/phpsolutions/images/');
    define('THUMBS_DIR', 'C:/upload_test/thumbs/');
    define('MAX_WIDTH', 120);
    define('MAX_HEIGHT', 90);
}
```

The new code defines four constants: the folder containing the original images, the folder where the resized images are to be stored, and the maximum width and height you want the thumbnails to be. You could use ordinary variables, but defining constants at the start of a script makes it easy to identify default values and change them at a later stage. Note that the folder pathnames must end with a trailing slash.

*If you're using a remote server or a Mac, replace the pathnames just shown with the correct paths to your `images` and `thumbs` folders. The download files also use the pathnames for a Windows local testing environment, so you need to make the changes there, too.*

2. When the form is submitted, the `pix` element of the `$_POST` array contains the name of the image you want to resize. PHP needs to know the full path to the image, so combine the value of the `SOURCE_DIR` constant with `$_POST['pix']`, and assign it to a shorter variable like this (the code goes inside the conditional statement immediately after the four constants inserted in the previous step):

```
// get image name and build full pathname
if (!empty($_POST['pix'])) {
    $original = SOURCE_DIR.$_POST['pix'];
}
else {
    $original = NULL;
}
```

The static option of the drop-down menu has no value, so you need to check that `$_POST['pix']` isn't empty. If it is, `$original` is set to `NULL` to prevent the rest of the script from going ahead.

3. Next comes the script to calculate the scaling ratio. Insert the following code after the code in the previous step (still inside the original conditional statement):

```
// abandon processing if no image selected
if (!$original) {
    echo 'No image selected';
}
// otherwise resize the image
else {
    // begin by getting the details of the original
    list($width, $height, $type) = getimagesize($original);
    // calculate the scaling ratio
    if ($width <= MAX_WIDTH && $height <= MAX_HEIGHT) {
        $ratio = 1;
    }
    elseif ($width > $height) {
        $ratio = MAX_WIDTH/$width;
    }
    else {
        $ratio = MAX_HEIGHT/$height;
    }
    echo "Image selected: $original<br />";
    echo "Original width: $width<br />Original height: $height<br />";
    echo "Image type: $type<br />Scaling ratio: $ratio";
}
```

Although there should never be any output ahead of the `DOCTYPE` declaration, `echo` is being used here simply for testing purposes and will be removed later. When building scripts, it's always a good idea to display the result of a calculation or conditional statement, as it helps confirm you're getting the expected results.

The following line of code needs a little explanation:

```
list($width, $height, $type) = getimagesize($original);
```

As you saw in Chapter 4, `getimagesize()` returns an array containing four elements. On that occasion, we were interested only in the fourth element: a string containing the width and height of an image, ready to insert into an `<img>` tag. This time, it's the first three elements we want: the width, height, and image type.

The `list()` construct lets you assign array elements directly to variables. The array elements are assigned to the variables in the same order. So, the first variable (`$width`) gets the first element of the array produced by `getimagesize($original)`—in other words, the image's width. The second variable (`$height`) gets the height of the image, and so on. If you pass fewer variables to `list()` than the number of array elements, any surplus ones are ignored.

The calculation of the scaling ratio is a simple arithmetic calculation. If the width and height of the original image are smaller or equal to the maximum, you don't want to scale the image. So the ratio is set to 1. Otherwise, you divide the maximum by the larger of the two dimensions. If the image is square, the ratio is determined by dividing the maximum height by the height of the original.

4. Save `create_thumb.php` and load it in a browser. Click Create thumbnail without selecting an image. You should see No image selected at the top of the screen. Then pick an image from the drop-down menu, and test the page again. You should see something like the screenshot to the right.

Try several different images. The scaling ratio should change for images of different dimensions. All the photos in the images folder are JPEG files, so Image type should always be 2.

Compare your code with `create_thumb02.php` in the download files, if necessary.

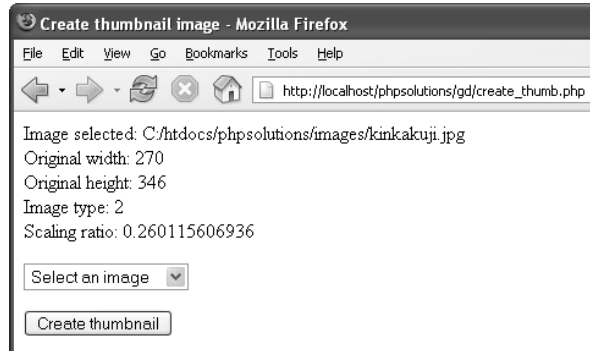
The `getimagesize()` function returns the image type as a number. You can find a full list at [www.php.net/manual/en/function.getimagesize.php](http://www.php.net/manual/en/function.getimagesize.php). The ones of interest to web developers are the first three, as follows:

- 1 GIF
- 2 JPEG
- 3 PNG

The GD image functions handle each type of image differently, so these numbers will be used to create the script's branching logic.

### PHP Solution 8-2: Creating a scaled-down copy

Continue working with the same file. Alternatively, use `create_thumb02.php` from the download files. The finished script for this section is in `create_thumb03.php`.



1. You no longer need to display the results onscreen, so change the first echo command in step 3 of the previous section like this:

```
if (!$original) {
    $result = 'No image selected';
}
```

2. Delete the three echo commands at the end of the code in step 3, and replace them with the following code:

```
    else {
        $ratio = MAX_HEIGHT/$height;
    }
    // strip the extension off the image filename
    $imagentypes = array('/\.gif$/','/\.jpg$/','/\.jpeg$/','/\.png$/');
    $name = preg_replace($imagentypes, '', basename($original));
}
}
```

*Don't forget that all the code needs to go inside the original conditional statement that makes sure the script runs only when the form has been submitted. I've included the closing curly brace of that statement in the preceding code as a reminder.*

The first new line of code creates an array of regular expressions to identify the following filename extensions: .gif, .jpg, .jpeg, and .png. The next line uses `basename()` to extract the filename and passes it to `preg_replace()`, which searches the `$imagentypes` array for a match and replaces it with nothing. Let's say `$original` contains the following pathname:

```
C:/htdocs/phpsolutions/images/kinkakuji.jpg
```

By passing it to `basename()`, it becomes this:

```
kinkakuji.jpg
```

Finally, .jpg is removed, leaving you with this:

```
kinkakuji
```

This value is stored in `$name` and can be used to build the name of the resized image.

3. As explained earlier, the first step in working with an image in PHP is to create an image resource in memory. To create a scaled-down copy, you need two image resources: one for the original image and another for the thumbnail image. Let's begin with the original image.

The function used to create an image resource from an existing image depends on the file type. Since you stored that information in `$type`, you can use a switch statement (see "Using the switch statement for decision chains" in Chapter 3) to

select the appropriate function. Insert the following code immediately after the code in the previous step:

```
$name = preg_replace($imagetypes, '', basename($original));
// create an image resource for the original
switch($type) {
  case 1:
    $source = @ imagecreatefromgif($original);
    if (!$source) {
      $result = 'Cannot process GIF files. Please use JPEG or PNG.';
    }
    break;
  case 2:
    $source = imagecreatefromjpeg($original);
    break;
  case 3:
    $source = imagecreatefrompng($original);
    break;
  default:
    $source = NULL;
    $result = 'Cannot identify file type.';
}
}
```

The switch statement checks the number stored in \$type and creates an image resource called \$source using the correct function for the file type. All servers should support imagecreatefromjpeg() and imagecreatefrompng(), but a server using an older version of GD might not support imagecreatefromgif(). That's why I have used the error control operator (see "Preventing errors when an include file is missing" in Chapter 4) if \$type is 1 (a GIF file). If the server can't handle GIF files, \$source will be false, so a suitable error message is stored in \$result. A different error message is created if \$type is not 1, 2, or 3.

4. After making sure that the image resource for the original is OK, you can go ahead and create the thumbnail. Insert the following code immediately after the switch statement from the previous step:

```
$result = 'Cannot identify file type.';
}
// make sure the image resource is OK
if (!$source) {
  $result = 'Problem copying original';
}
else {
  // calculate the dimensions of the thumbnail
  $thumb_width = round($width * $ratio);
  $thumb_height = round($height * $ratio);
  // create an image resource for the thumbnail
  $thumb = imagecreatetruecolor($thumb_width, $thumb_height);
```

```

        // create the resized copy
        // save the resized copy
        // remove the image resources from memory
    }
}
}

```

If `$source` is false, there must be a problem with copying the original, so there's no point in continuing. However, if a valid image resource exists, the `else` statement is executed. It begins by multiplying the width and height of the original by the scaling ratio. Because the dimensions of an image must be integers, the calculation is passed to `round()`, which returns the nearest whole number.

Then you need to create the image resource for the resized copy. This is done by passing the width and height of the thumbnail to `imagecreatetruecolor()` and storing the resource as `$thumb`.

Just three more steps remain, as indicated by the three comments at the end of the new code. Before moving on to them, let's pause to see how the copy is actually created.

The function that creates a resized copy of an image is `imagecopyresampled()`, which takes—wait for it—ten arguments! While this sounds horrendous, the arguments fall into five pairs as follows:

- References to the two image resources—copy first, original second
- The x and y coordinates of where to position the top-left corner of the copied image
- The x and y coordinates of the top-left corner of the original
- The width and height of the copy
- The width and height of the area to copy from the original

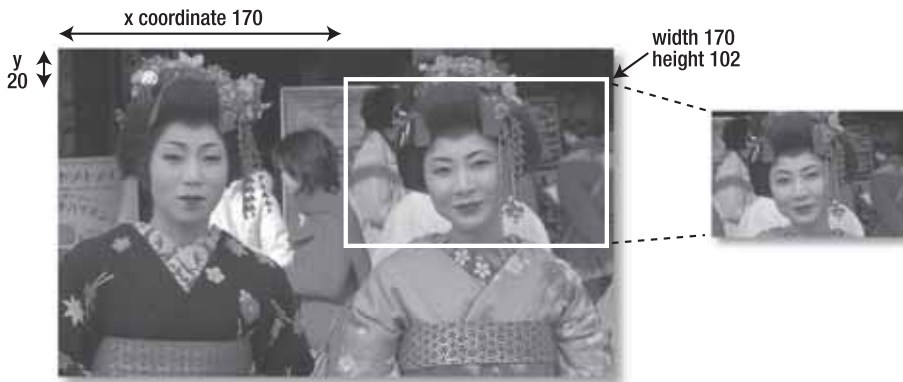
The only time you need to worry about the x and y coordinates is when you want to extract a specific area, rather than copy the whole image. The coordinates are measured in pixels from the top left of the image. Figure 8-1 shows the effect of the following code:

```

imagecopyresampled($thumb, $source, 0, 0, 170, 20, $thumb_width, ➤
$thumb_height, 170, 102);

```

The x coordinate of the original image is set at 170 pixels from the top left, and the y coordinate is at 20 pixels. By setting the width and height of the original to 170 and 102, respectively, PHP copies only the area outlined in white. Although this is impressive, you can probably already see a problem: you need to view the image first to find the best area to extract and get the correct coordinates. Although you could use a pixel ruler, it's messy. You get a much more satisfactory result in a graphics program, such as Photoshop, where you can crop and rescale the image quickly and accurately.



**Figure 8-1.** The x and y coordinates can be used to extract part of an image.

Consequently, when using PHP to create a thumbnail, the x and y coordinates aren't really relevant. You simply set all four coordinates to the top-left corner (0), and use the actual width and height of the original image. This makes a thumbnail of the entire image. So let's get back to the code.

5. Insert the following line of code immediately beneath the first of the three comments that you inserted at the end of the previous step:

```
// create the resized copy
imagecopyresampled($thumb, $source, 0, 0, 0, 0, $thumb_width, ↗
$thumb_height, $width, $height);
```

Note that you don't need to assign the result of `imagecopyresampled()` to a variable; `$thumb` now contains the scaled-down image, but you still need to save it to a file.

6. The functions that save an image to file need to know the file type. So this means using another switch statement to select the appropriate function: `imagegif()`, `imagejpeg()`, or `imagepng()`. Each function takes the following two arguments:
  - The image resource being held in memory
  - The pathname of the file you want to save the image to

You can build the pathname with `THUMBS_DIR`, followed by `$name` (the original filename minus the extension), plus `_thb.gif`, `_thb.jpg`, or `_thb.png`, as appropriate. This results in the thumbnail for `kinkakuji.jpg` being saved in the `thumbs` subfolder as `kinkakuji_thb.jpg`.

The function that creates JPEG files takes a third, optional argument: an integer between 0 and 100 to indicate the quality of the image. JPEG compresses the image, so a lower number produces a smaller file size, but of lower quality. If you omit the third argument, the default is 75. Except when saving a GIF in JPEG format, it's a good idea to specify 100. You can always reduce the quality later if the file is too big, but you can't restore picture quality once it's been reduced.

Place this code under the second comment:

```
// save the resized copy
switch($type) {
  case 1:
    if (function_exists('imagegif')) {
      $success = imagegif($thumb, THUMBS_DIR.$name.'_thb.gif');
      $thumb_name = $name.'_thb.gif';
    }
    else {
      $success = imagejpeg($thumb, THUMBS_DIR.$name.'_thb.jpg', 50);
      $thumb_name = $name.'_thb.jpg created';
    }
    break;
  case 2:
    $success = imagejpeg($thumb, THUMBS_DIR.$name.'_thb.jpg', 100);
    $thumb_name = $name.'_thb.jpg created';
    break;
  case 3:
    $success = imagepng($thumb, THUMBS_DIR.$name.'_thb.png');
    $thumb_name = $name.'_thb.png created';
}
if ($success) {
  $result = "$thumb_name created";
}
else {
  $result = 'Problem creating thumbnail';
}
```

As with the earlier switch statement, you need to check whether the server supports GIF. Even if the server can read GIF files, it might not be able to create them, so case 1 begins by using `function_exists()` to establish if it's safe to use `imagegif()`. If it is, `imagegif()` is used to save the thumbnail to file. If GIF creation isn't supported, the else clause uses `imagejpeg()` to save it as a JPEG file with a quality of 50.

*Note that `function_exists()` takes the name of the function as a string without the final parentheses like this:*

```
if (function_exists('imagegif')) // RIGHT
if (function_exists(imagegif())) // WRONG
```

The functions that save the image to file return a Boolean true or false, which is stored in `$success`. A message reporting the outcome is stored in `$result`.

7. All that remains is to remove from memory the two image resources that you've been working with. Place this code under the final comment:

```
// remove the image resources from memory
imagedestroy($source);
imagedestroy($thumb);
```

In spite of its destructive name, `imagedestroy()` has no effect on the original image, nor on the thumbnail that's just been saved to file. The function simply frees up the server memory by destroying the image resources required during processing.

8. Before testing the page, you need to add some code just after the opening `<body>` tag to display the message reporting the outcome like this:

```
<body>
<?php
if (isset($result)) {
    echo "<p>$result</p>";
}
?>
<form id="form1" name="form1" method="post" action="">
```

9. Save `create_thumb.php` and load it in a browser. Select an image from the drop-down menu and click `Create thumbnail`. If all goes well, there should be a scaled-down version of the image you chose in the `thumbs` subfolder of `upload_test`. Check your code, if necessary, with `create_thumb03.php` in the download files.

## Resizing an image automatically on upload

Now that you have a script that creates a thumbnail from a larger image, it takes only a few minor changes to merge it with the file upload script from Chapter 6. Rather than build the entire script in a single page, this is a good opportunity to use a PHP include (includes were covered in Chapter 4).

8

### PHP Solution 8-3: Merging the image upload and resizing scripts

The starting point for this PHP Solution is `create_thumb.php` from the preceding section, together with `upload.php` from Chapter 6. Alternatively, use `create_thumb03.php` and `upload_thumb01.php` from the download files for this chapter. The finished scripts are in `create_thumb.inc.php` and `upload_thumb02.php`.

1. In `create_thumb.php`, select the entire PHP block above the `DOCTYPE` declaration. Copy the selected code to your computer clipboard, and paste it inside a blank PHP page. The new page should contain PHP script only; you don't need a `DOCTYPE` or `XHTML` skeleton. Save the page in the `includes` folder as `create_thumb.inc.php`.
2. Remove the comment on line 2 together with the conditional statement that surrounds the script (don't forget the closing curly brace just before the closing PHP tag). You should be left with the following:

```
<?php
// define constants
define('SOURCE_DIR', 'C:/htdocs/phpsolutions/images/');
define('THUMBS_DIR', 'C:/upload_test/thumbs/');
define('MAX_WIDTH', 120);
define('MAX_HEIGHT', 90);
```

```

// get image name and build full pathname
if (!empty($_POST['pix'])) {
    $original = SOURCE_DIR.$_POST['pix'];
}
else {
    $original = NULL;
}
// abandon processing if no image selected
if (!$original) {
    $result = 'No image selected';
}
// otherwise resize the image
else {
    // begin by getting the details of the original
    list($width, $height, $type) = getimagesize($original);
    // calculate the scaling ratio
    if ($width <= MAX_WIDTH && $height <= MAX_HEIGHT) {
        $ratio = 1;
    }
    elseif ($width > $height) {
        $ratio = MAX_WIDTH/$width;
    }
    else {
        $ratio = MAX_HEIGHT/$height;
    }
    // strip the extension off the image filename
    $imagetypes = array('/\.gif$/', '/\.jpg$/', '/\.jpeg$/', '/\.png$/');
    $name = preg_replace($imagetypes, '', basename($original));

    // create an image resource for the original
    switch($type) {
        case 1:
            $source = @ imagecreatefromgif($original);
            if (!$source) {
                $result = 'Cannot process GIF files. Please use JPEG or PNG.';
            }
            break;
        case 2:
            $source = imagecreatefromjpeg($original);
            break;
        case 3:
            $source = imagecreatefrompng($original);
            break;
        default:
            $source = NULL;
            $result = 'Cannot identify file type.';
    }
    // make sure the image resource is OK
    if (!$source) {

```

```

    $result = 'Problem copying original';
  }
else {
  // calculate the dimensions of the thumbnail
  $thumb_width = round($width * $ratio);
  $thumb_height = round($height * $ratio);
  // create an image resource for the thumbnail
  $thumb = imagecreatetruecolor($thumb_width, $thumb_height);
  // create the resized copy
  imagecopyresampled($thumb, $source, 0, 0, 0, 0, $thumb_width,
$thumb_height, $width, $height);
  // save the resized copy
  switch($type) {
    case 1:
      if (function_exists('imagegif')) {
        $success = imagegif($thumb, THUMBS_DIR.$name.'_thb.gif');
        $thumb_name = $name.'_thb.gif';
      }
      else {
        $success = imagejpeg($thumb, THUMBS_DIR.$name.'_thb.jpg',50);
        $thumb_name = $name.'_thb.jpg';
      }
      break;
    case 2:
      $success = imagejpeg($thumb, THUMBS_DIR.$name.'_thb.jpg', 100);
      $thumb_name = $name.'_thb.jpg';
      break;
    case 3:
      $success = imagepng($thumb, THUMBS_DIR.$name.'_thb.png');
      $thumb_name = $name.'_thb.png';
  }
  if ($success) {
    $result = "$thumb_name created";
  }
  else {
    $result = 'Problem creating thumbnail';
  }
  // remove the image resources from memory
  imagedestroy($source);
  imagedestroy($thumb);
}
}
?>

```

As the script now stands, it looks for the name of an image submitted from a form as `$_POST['pix']`, and located on the server in whatever you have defined as `SOURCE_DIR`. To create a thumbnail from an uploaded image, you need to adapt the script so that it processes the temporary upload file.

If you cast your mind back to Chapter 6, PHP stores an upload file in a temporary location until you move it to its target location. This temporary file is accessed using the `tmp_name` element of the `$_FILES` superglobal array and is discarded when the script ends. Instead of moving the temporary file to the upload folder, you can adapt the script in `create_thumb.inc.php` to resize the image, and save the scaled-down version instead.

3. The form in `upload.php` uses `image` as the name attribute of the file upload field, so the original image (referred to as `$original`) is now in `$_FILES['image']['tmp_name']`. Change the opening section of the code like this (new code is in bold):

```
// define constants
define('THUMBS_DIR', 'C:/upload_test/thumbs/');
define('MAX_WIDTH', 120);
define('MAX_HEIGHT', 90);

// process the uploaded image
if (is_uploaded_file($_FILES['image']['tmp_name'])) {
    $original = $_FILES['image']['tmp_name'];
    // begin by getting the details of the original
    list($width, $height, $type) = getimagesize($original);
```

This removes the definition of `SOURCE_DIR`, which is no longer needed, and simplifies the original `if... else` statements at the beginning of the script. The code in `upload.php` takes care of checking that a file has been selected, so all that's needed here is to use `is_uploaded_file()` to check that the temporary file is a genuine upload and to assign it to `$original`.

*If you ever had any doubts, this should convince you just how useful variables are. From this point on, the script treats the temporary upload file in exactly the same way as a file already on the server. The remaining steps also demonstrate the value of recycling code.*

4. Save `create_thumb.inc.php`. The rest of the changes are made in the upload file.
5. Open `upload.php` from Chapter 6 and save it as `upload_thumb.php`.
6. Locate the following section of code in `upload_thumb.php` (it should be around lines 32 through 60):

```
if ($sizeOK && $typeOK) {
    switch($_FILES['image']['error']) {
        case 0:
            // $username would normally come from a session variable
            $username = 'davidp';
            // if the user's subfolder doesn't exist yet, create it
            if (!is_dir(UPLOAD_DIR.$username)) {
                mkdir(UPLOAD_DIR.$username);
            }
            // check if a file of the same name has been uploaded
            if (!file_exists(UPLOAD_DIR.$username.'/'.$file)) {
```

```

        // move the file to the upload folder and rename it
        $success = move_uploaded_file($_FILES['image']['tmp_name'], ↵
        UPLOAD_DIR.$username.'/'.$file);
    }
    else {
        // get the date and time
        ini_set('date.timezone', 'Europe/London');
        $now = date('Y-m-d-His');
        $success = move_uploaded_file($_FILES['image']['tmp_name'], ↵
        UPLOAD_DIR.$username.'/'.$now.$file);
    }
    if ($success) {
        $result = "file uploaded successfully";
    }
    else {
        $result = "Error uploading $file. Please try again.";
    }
    break;

```

7. Change it to this:

```

if ($sizeOK && $typeOK) {
    switch($_FILES['image']['error']) {
        case 0:
            include('../includes/create_thumb.inc.php');
            break;

```

That's it! Save `upload_thumb.php` and test it by selecting an image from your local file system: a scaled-down copy will be created in the `thumbs` subfolder of `upload_test` (see Figure 8-2).

Check your code, if necessary, with `create_thumb.inc.php` and `upload_test02.php` in the download files.

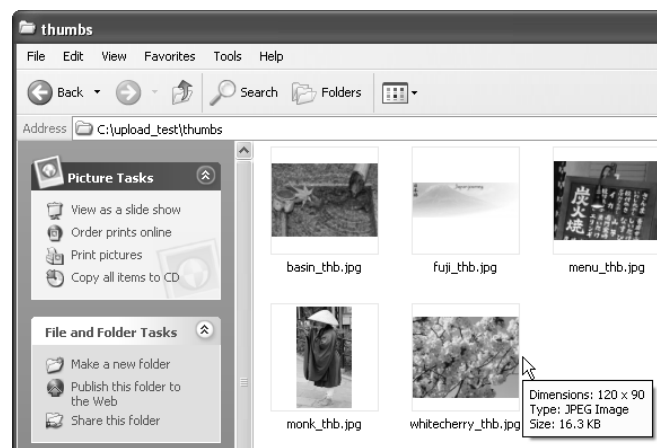


Figure 8-2. A  $400 \times 300$  pixel image has been automatically resized and renamed on upload.

To understand what has happened, cast your mind back to Chapter 6. The switch statement checks the value of `$_FILES['image']['error']`. If it's 0, it means that the upload succeeded. The original script moved the temporary upload file to its target destination. The `include` command simply replaces that part of the script with the code that creates the thumbnail.

## Further improvements

You now have a powerful mini-application that automatically resizes images on upload, but what if you want to preserve the original image as well? Nothing could be simpler. The page containing the upload form already defines the upload folder as `UPLOAD_DIR`, so you simply need to move the temporary upload file (currently referred to as `$original`) with `move_uploaded_file()`.

### PHP Solution 8-4: Saving the uploaded original and scaled-down version

Continue working with the same files. Alternatively, use `create_thumb.inc.php` and `upload_thumb02.php` from the download files. The finished scripts are in `create_both.inc.php` and `upload_both.php`.

1. Open `upload_thumb.php` and save a copy as `upload_both.php`.
2. In `upload_both.php`, locate the line that includes the script that creates the scaled-down image. It should be around line 35, and looks like this:

```
include('../includes/create_thumb.inc.php');
```

Change it like this and save the page:

```
include('../includes/create_both.inc.php');
```

3. Open `create_thumb.inc.php` and save a copy in the `includes` folder as `create_both.inc.php`.
4. In `create_both.inc.php`, locate the section of code that strips the extension from the filename (around line 22), and insert the new code highlighted in bold:

```
// strip the extension off the image filename
$imagetypes = array('/\.gif$/','/\.jpg$/','/\.jpeg$/','/\.png$/');
$name = preg_replace($imagetypes, '', ↵
basename($_FILES['image']['name']));
```

```
// move the temporary file to the upload folder
$moved = @ move_uploaded_file($original, ↵
UPLOAD_DIR.$_FILES['image']['name']);
if ($moved) {
    $result = $_FILES['image']['name'].' successfully uploaded; ';
    $original = UPLOAD_DIR.$_FILES['image']['name'];
}
else {
```

```
$result = 'Problem uploading '.$_FILES['image']['name'].'. ';
}
```

```
// create an image resource for the original
```

The new code moves the temporary upload file to the upload folder and saves it with its original name. The `move_uploaded_file()` function returns a Boolean `true` or `false`, so by assigning the result to `$moved`, you can tell whether the operation is successful. If it is, a suitable message is created, and the pathname of the uploaded file is reassigned to `$original`. *This is very important*, because `move_uploaded_file()` immediately discards the temporary uploaded file. So, from this point onward, the original file is now the one that has just been saved on the server.

If `$moved` is `false`, there's no point in reassigning the value of `$original`, which still points to the temporary upload file. This means you still have a chance of creating the thumbnail, even if the main upload fails. I've inserted the error control operator (`@`) in front of `move_uploaded_file()` to prevent the display of PHP error messages, so it's important to create a custom error message indicating what the problem is.

5. The outcome of the upload operation uses the same variable, `$result`, as the section of the script that creates the resized image, so you need to make sure that the second outcome is added to the first. Do this with the combined concatenation operator (`.=`) toward the end of the script, by inserting a period in front of the equal sign like this:

```
if ($success) {
    $result .= "$thumb_name created";
}
else {
    $result .= 'Problem creating thumbnail';
}
```

*As it stands, the script gives you the chance to salvage at least part of the operation if the main upload fails. If you don't want a thumbnail without the main image, move the last four lines of new code in step 4 immediately below the code in step 5. This brings the thumbnail creation script inside the first half of the conditional statement, so it runs only if `$moved` is `true`.*

6. Save `create_both.inc.php`, and load `upload_both.php` into a browser. Test it by selecting an image on your local computer and clicking `Upload`. The original image should be copied to the `upload_test` folder and a scaled-down version to the `thumbs` subfolder.

You may be wondering why I inserted the new code in step 4 in that particular location, because it doesn't really matter when you move the uploaded file, as long as the script can create an image resource from it. The answer is because the script currently overwrites existing images of the same name. For a really robust solution, you need to assign a unique

name to each file as it's uploaded. By placing `move_uploaded_file()` at this point, you can use the value of `$name` to generate a unique name for the uploaded file and its thumbnail.

Rather than show you how to do it step by step, I'll just give you a few hints. The `getNextFilename()` function from the previous chapter automatically generates a new filename. It takes three arguments: the target folder (directory), the filename's prefix, and the file type. The target directory is `UPLOAD_DIR`, the filename's prefix is stored in `$name`, and the file type is stored in `$type`. However, `$type` is currently a number, so you need to convert it to a string. If you store the new name in `$newName`, you can use it in combination with `basename()` to build the name for the thumbnail so that the original image and thumbnail have the same number. Refer back to PHP Solution 4-3 for an explanation of how to use `basename()`.

The changes involved are quite simple and involve fewer than 20 lines of code. The solution is in `upload_both_new.php` and `create_both_new.inc.php` in the download files. The new code is clearly marked and commented.

## Transferring your test files to a remote server

If you have been testing these files locally, the only changes that you need to make when deploying them on a remote server are to the definitions of `UPLOAD_DIR` and `THUMBS_DIR`. Use a fully qualified path to each folder (directory). Don't forget that the necessary read, write, and execute permissions need to be set on the upload folders. Also make sure that the path to any include files reflects your site structure.

Change the values of `MAX_HEIGHT` and `MAX_WIDTH` if you want the resized images to be larger or smaller than  $120 \times 90$  pixels.

## Summary

Although this is a relatively short chapter, it covers a lot of ground and brings together techniques from Chapters 4, 6, and 7, in combination with the PHP image manipulation functions. To get the most out of working with PHP, it's important to understand the flow of a script so that you can incorporate solutions from other scripts. It would be a major project to attempt to build from scratch a form that uploads an image, makes a scaled-down copy, and gives both of them new names. However, breaking the task down into discrete sections, as done here, makes it a lot easier. It also gives you the opportunity to reuse code from one project in another, saving time and effort.

There are many other things you can do with the GD extension, including adding dynamic text to images and generating bar charts. For more details, take a look at Chapter 8 of *PHP 5 Recipes: A Problem-Solution Approach* by Lee Babin and others (Apress, ISBN: 1-59059-509-2).

