Thumbnails in PHP erstellen

== Thumbnail Klasse ==

Quellcode

```
<?php
/**
* Generates a thumbnail of given source file image.
*
* @license GNU Lesser General Public License <http://opensource.org/licenses/lgpl-license.php>
*/
class Thumbnail {
/**
* path to source file
*
* @var string
*/
protected $sourceFile = '';
/**
* maximum image width
*
* @var integer
*/
protected $maxWidth = 100;
/**
* maximum image height
*
* @var integer
*/
protected $maxHeight = 100;
/**
* true, to show source information in thumbnail
*
* @var boolean
*/
protected $appendSourceInfo = false;
/**
* true, to prefer embedded thumbnails
*
* @var boolean
*/
protected $useEmbedded = true;
/**
* true, to generate quadratic thumbnails
*
* @var boolean
*/
protected $quadratic = false;
/**
* mime type of the thumbnail
*
* @var string
*/
protected $mimeType = '';
/**
* name of the source image
*
```

```
61.  * @var string
62.  */
63.  protected $sourceName = '';
64.
65.  /**
66.  * width of the source image
67.  *
68.  * @var integer
69.  */
70.  protected $sourceWidth = 0;
71.
72.  /**
73.  * height of the source image
74.  *
75.  * @var integer
76.  */
77.  protected $sourceHeight = 0;
78.
79.  /**
80.  * file size of the source image
81.  *
82.  * @var integer
83.  */
84.  protected $sourceSize = 0;
85.
86.  /**
87.  * height of the source information
88.  *
89.  * @var integer
90.  */
91.  protected static $sourceInfoLineHeight = 16;
92.
93.  /**
94.  * image type of the source image
95.  *
96.  * @var integer
97.  */
98.  protected $imageType = 0;
99.
100.  /**
101.  * Creates a new Thumbnail object.
102.  *
103.  * @param string $sourceFile
104.  */
105.  public function __construct($sourceFile) {
106.  $this->sourceFile = $sourceFile;
107.  }
108.
109.  /**
110.  * setup dimenstions
111.  *
112.  * @param boolean $appendSourceInfo
113.  */
114.  public function setup($appendSourceInfo = false, $sourceName = null, $useEmbedded = true, $quadratic = false) {
115.  $this->appendSourceInfo = $appendSourceInfo;
116.  $this->useEmbedded = $useEmbedded;
117.  $this->quadratic = $quadratic;
118.  if ($this->appendSourceInfo) {
119.  // get source info
120.  if ($sourceName != null) $this->sourceName = $sourceName;
121.  else $this->sourceName = basename($this->sourceFile);
122.  list($this->sourceWidth, $this->sourceHeight, $type) = @getImageSize($this->sourceFile);
123.  $this->sourceSize = @filesize($sourceFile);
124.  }
125.  }
```

```
/**
 * Creates a thumbnail picture (jpg/png) of a big image
 *
 * @param boolean $rescale
 * @return string thumbnail
 */
protected function makeThumbnail($rescale = false) {
list($width, $height, $this->imageType) = @getImageSize($this->sourceFile);
// check image size
if ($this->checkSize($width, $height, $rescale)) {
return false;
}
// try to extract the embedded thumbnail first (faster)
$thumbnail = false;
if (!$rescale && $this->useEmbedded) {
$thumbnail = $this->extractEmbeddedThumbnail();
}
if (!$thumbnail) {
// calculate uncompressed filesize
// and cancel to avoid a memory_limit error
$memoryLimit = ini_get('memory_limit');
if ($memoryLimit != '') {
$memoryLimit = substr($memoryLimit, 0, -1) * 1024 * 1024;
$fileSize = $width * $height * ($this->imageType == 3 ? 4 : 3);
if (($fileSize * 2.1) + memory_get_usage() > ($memoryLimit)) {
return false;
}
}
// calculate new picture size
$x = $y = 0;
if ($this->quadratic) {
$newWidth = $newHeight = $this->maxWidth;
if ($this->appendSourceInfo) $newHeight -= self::$sourceInfoLineHeight * 2;
if ($width > $height) {
$x = ceil(($width - $height) / 2);
$width = $height;
}
else {
$y = ceil(($height - $width) / 2);
$height = $width;
}
}
else {
$maxHeight = $this->maxHeight;
if ($this->appendSourceInfo) $maxHeight -= self::$sourceInfoLineHeight * 2;
if ($this->maxWidth / $width < $maxHeight / $height) {
$newWidth = $this->maxWidth;
$newHeight = round($height * ($newWidth / $width));
}
else {
$newHeight = $maxHeight;
$newWidth = round($width * ($newHeight / $height));
}
}
// resize image
$imageResource = false;
// jpeg image
if ($this->imageType == 2 && function_exists('imagecreatefromjpeg')) {
```

```
192.  $imageResource = @imageCreateFromJPEG($this->sourceFile);
193.  }
194.  // gif image
195.  if ($this->imageType == 1 && function_exists('imagecreatefromgif')) {
196.  $imageResource = @imageCreateFromGIF($this->sourceFile);
197.  }
198.  // png image
199.  if ($this->imageType == 3 && function_exists('imagecreatefrompng')) {
200.  $imageResource = @imageCreateFromPNG($this->sourceFile);
201.  }
202.  // could not create image
204.  if (!$imageResource) {
205.  return false;
206.  }
207.  // resize image
209.  if (function_exists('imageCreateTrueColor') && function_exists('imageCopyResampled')) {
210.  $imageNew = @imageCreateTrueColor($newWidth, $newHeight);
211.  imageAlphaBlending($imageNew, false);
212.  @imageCopyResampled($imageNew, $imageResource, 0, 0, $x, $y, $newWidth, $newHeight, $width, $height);
213.  imageSaveAlpha($imageNew, true);
214.  }
215.  else if (function_exists('imageCreate') && function_exists('imageCopyResized')) {
216.  $imageNew = @imageCreate($newWidth, $newHeight);
217.  imageAlphaBlending($imageNew, false);
218.  @imageCopyResized($imageNew, $imageResource, 0, 0, $x, $y, $newWidth, $newHeight, $width, $height);
219.  imageSaveAlpha($imageNew, true);
220.  }
221.  else return false;
223.  // create thumbnail
224.  ob_start();
226.  if ($this->imageType == 1 && function_exists('imageGIF')) {
227.  @imageGIF($imageNew);
228.  $this->mimeType = 'image/gif';
229.  }
230.  else if (($this->imageType == 1 || $this->imageType == 3) && function_exists('imagePNG')) {
231.  @imagePNG($imageNew);
232.  $this->mimeType = 'image/png';
233.  }
234.  else if (function_exists('imageJPEG')) {
235.  @imageJPEG($imageNew, '', 90);
236.  $this->mimeType = 'image/jpeg';
237.  }
238.  else {
239.  return false;
240.  }
242.  @imageDestroy($imageNew);
243.  $thumbnail = ob_get_contents();
244.  ob_end_clean();
245.  }
247.  if ($thumbnail && $this->appendSourceInfo && !$rescale) {
248.  $thumbnail = $this->appendSourceInfo($thumbnail);
249.  }
251.  return $thumbnail;
252.  }
254.  /**
255.  * Appends information about the source image to the thumbnail.
256.  *
257.  * @param string $thumbnail
```

```
258.  * @return string
259.  */
260.  protected function appendSourceInfo($thumbnail) {
261.  if (!function_exists('imageCreateFromString') || !function_exists('imageCreateTrueColor')) {
262.  return $thumbnail;
263.  }
264.  $imageSrc = imageCreateFromString($thumbnail);
265.  // get image size
266.  $width = imageSX($imageSrc);
267.  $height = imageSY($imageSrc);
268.  // increase height
269.  $heightDst = $height + self::$sourceInfoLineHeight * 2;
270.  // create new image
271.  $imageDst = imageCreateTrueColor($width, $heightDst);
272.  imageAlphaBlending($imageDst, false);
273.  // set background color
274.  $background = imageColorAllocate($imageDst, 102, 102, 102);
275.  imageFill($imageDst, 0, 0, $background);
276.  // copy image
277.  imageCopy($imageDst, $imageSrc, 0, 0, 0, 0, $width, $height);
278.  imageSaveAlpha($imageDst, true);
279.  // get font size
280.  $font = 2;
281.  $fontWidth = imageFontWidth($font);
282.  $fontHeight = imageFontHeight($font);
283.  $fontColor = imageColorAllocate($imageDst, 255, 255, 255);
284.  // write source info
285.  if($maxWidth) $this->maxWidth = $maxWidth;
286.  if($maxHeight) $this->maxHeight = $maxHeight;
287.  $line1 = $this->sourceName;
288.  // imageString supports only ISO-8859-1 encoded strings
289.  // if (CHARSET != 'ISO-8859-1') {
290.  // $line1 = StringUtil::convertEncoding(CHARSET, 'ISO-8859-1', $line1);
291.  // }
292.  // truncate text if necessary
293.  $maxChars = floor($width / $fontWidth);
294.  if (strlen($line1) > $maxChars) {
295.  $line1 = $this->truncateSourceName($line1, $maxChars);
296.  }
297.  $line2 = $this->sourceWidth.'x'.$this->sourceHeight;
298.  // write line 1
299.  // calculate text position
300.  $textX = 0;
301.  $textY = 0;
302.  if ($fontHeight < self::$sourceInfoLineHeight) {
303.  $textY = intval(round((self::$sourceInfoLineHeight - $fontHeight) / 2));
304.  }
305.  if (strlen($line1) * $fontWidth < $width) {
306.  $textX = intval(round(($width - strlen($line1) * $fontWidth) / 2));
307.  }
308.  imageString($imageDst, $font, $textX, $height + $textY, $line1, $fontColor);
309.  // write line 2
310.  // calculate text position
311.  $textX = 0;
312.  $textY = 0;
313.  if ($fontHeight < self::$sourceInfoLineHeight) {
314.  $textY = self::$sourceInfoLineHeight + intval(round((self::$sourceInfoLineHeight - $fontHeight) / 2));
315.  }
```

```
332. if (strlen($line2) * $fontWidth < $width) {
333. $textX = intval(round(($width - strlen($line2) * $fontWidth) / 2));
334. }
335. imageString($imageDst, $font, $textX, $height + $textY, $line2, $fontColor);
336. // output image
337. ob_start();
338. if ($this->imageType == 1 && function_exists('imageGIF')) {
339. @imageGIF($imageDst);
340. $this->mimeType = 'image/gif';
341. }
342. else if (($this->imageType == 1 || $this->imageType == 3) && function_exists('imagePNG')) {
343. @imagePNG($imageDst);
344. $this->mimeType = 'image/png';
345. }
346. else if (function_exists('imageJPEG')) {
347. @imageJPEG($imageDst, '', 90);
348. $this->mimeType = 'image/jpeg';
349. }
350. else {
351. return false;
352. }
353. @imageDestroy($imageDst);
354. $thumbnail = ob_get_contents();
355. ob_end_clean();
356. return $thumbnail;
357. }
358. /**
359. * Extracts the embedded thumbnail picture of a jpeg or tiff image
360. *
361. * @return string thumbnail
362. */
363. protected function extractEmbeddedThumbnail() {
364. if (!function_exists('exif_thumbnail')) {
365. return false;
366. }
367. $width = $height = $type = 0;
368. $thumbnail = @exif_thumbnail($this->sourceFile, $width, $height, $type);
369. if ($thumbnail && $type && $width && $height) {
370. // resize the extracted thumbnail again if necessary
371. // (normally the thumbnail size is set to 160px
372. // which is recommended in EXIF >2.1 and DCF)
373. $this->mimeType = image_type_to_mime_type($type);
374. if (!$this->checkSize($width, $height) && function_exists('sys_get_temp_dir')) {
375. // get temporary file name
376. $this->sourceFile = tempnam(sys_get_temp_dir(), 'Tux');
377. // create tmp file
378. $fh = fopen($this->sourceFile, "w");
379. fwrite($fh, $thumbnail);
380. fclose($fh);
381. unset($thumbnail, $tmpFile);
382. // resize tmp file again
383. return $this->makeThumbnail(true);
384. }
385. return $thumbnail;
386. }
387. return false;
388. }
389. /**
```

```
 * Checks the size of an image.
 */
protected function checkSize($width, $height, $rescale = true) {
$maxHeight = $this->maxHeight;
if ($this->appendSourceInfo && $rescale) {
$maxHeight -= self::$sourceInfoLineHeight * 2;
}
if ($width > $this->maxWidth || $height > $maxHeight) {
return false;
}
return true;
}
/**
 * Returns the mime type of the generated thumbnail.
 *
 * @return string
 */
public function getMimeType() {
return $this->mimeType;
}
/**
 * Truncates the given file name to needed length.
 *
 * @param string $name
 * @param string $maxChars
 * @return string
 */
protected static function truncateSourceName($name, $maxChars) {
$extension = '';
$lastPosition = strrpos($name, '.');
if ($lastPosition !== null) {
$extension = substr($name, $lastPosition);
$name = substr($name, 0, $lastPosition);
$maxChars -= strlen($extension);
}
return substr($name, 0, $maxChars - 3) . '...' . $extension;
}
/**
 *
 * @return boolean
 */
public function save($targetFile, $maxWidth = 0, $maxHeight = 0) {
$recover = array();
if($maxWidth) {
$recover['maxWidth'] = $this->maxWidth;
$this->maxWidth = $maxWidth;
}
if($maxHeight) {
$recover['maxHeight'] = $this->maxHeight;
$this->maxHeight = $maxHeight;
}
// get thumbnail
try {
$success = true;
if (($thumbnailData = $this->makeThumbnail())) {
// save thumbnail
$fh = fopen($targetFile, 'w');
fwrite($fh, $thumbnailData);
```

```
470.  unset($thumbnailData);
471.  fclose($fh);
472.  }
473.  }
474.  catch (Exception $e) {
475.  $success = false;
476.  }
477.  foreach($recover as $key => $val) {
478.  $this->$key = $val;
479.  }
480.  return $success;
481.  }
482.  protected static function sendLastModified($timestamp) {
483.  // Getting headers sent by the client.
484.  $headers = apache_request_headers();
485.  // Checking if the client is validating his cache and if it is current.
486.  if (isset($headers['If-Modified-Since']) && (strtotime($headers['If-Modified-Since']) >= $timestamp)) {
487.  // this timestamp can also be used to identify the user ;) see
488.  // Client's cache IS current, so we just respond '304 Not Modified'.
489.  header('Last-Modified: '.gmdate('D, d M Y H:i:s', $timestamp).' GMT', true, 304);
490.  exit;
491.  } else {
492.  // Image not cached or cache outdated, we respond '200 OK' and output the image.
493.  header('Last-Modified: '.gmdate('D, d M Y H:i:s', $timestamp).' GMT', true, 200);
494.  }
495.  }
496.  /**
497.  * send header and content for the picture
498.  */
499.  public function display($maxWidth = 100, $maxHeight = 100, $age = '3 MONTH') {
500.  $recover = array();
501.  if($maxWidth) {
502.  $recover['maxWidth'] = $this->maxWidth;
503.  $this->maxWidth = $maxWidth;
504.  }
505.  if($maxHeight) {
506.  $recover['maxHeight'] = $this->maxHeight;
507.  $this->maxHeight = $maxHeight;
508.  }
509.  $filemtime = @filemtime($this->sourceFile);
510.  $thumbnailData = $this->makeThumbnail();
511.  $size = strlen($thumbnailData);
512.  self::sendLastModified($filemtime);
513.  header('Accept-Ranges: bytes');
514.  header('Content-Length: '.$size);
515.  header('Content-Type: image/jpeg');
516.  header('Expires: '.gmdate('D, d M Y H:i:s', time() + strtotime($age)).' GMT');
517.  echo $thumbnailData;
518.  foreach($recover as $key => $val) {
519.  $this->$key = $val;
520.  }
521.  }
522.  }
523.  ?>
```

Alles anzeigen

== Beispiele ==

=== Bild in meheren Größen speichern ===

Quellcode

```
<?php
require_once('Thumbnail.php');
// this is the source
$thumb = new Thumbnail('demo/easy-coding.png');
// quadratic
$thumb->save('/tmp/easy-coding.png', 150, 150);
// 4:3 format
$thumb->save('/tmp/easy-coding.png', 100, 75);
?>
```

Alles anzeigen

== Bild "on the fly" anzeigen ==
Den Original Bildnamen könnt ihr zum Beispiel als $_GET Parameter oder mit mod_rewrite Regeln durchleiten. Im Beispiel wird die Vorschau für das Bild "easy-coding.png" erzeugt.
Der dritte Parameter der Methode display kann einen Zeitausdruck wie "3 month" enthalten, dann wird das Bild für drei Monate im Browser Cache des Benutzers gespeichert, ohne dass es erneut angefragt wird und damit Traffickosten verursacht. Allerdings bekommt der Benutzer auf diese Weise auch nicht von Änderungen mit, deswegen sollte der Intervall je nach Anwendung besser bei "1 day" liegen.

Quellcode

```
<?php
require_once('Thumbnail.php');
$thumb = new Thumbnail('demo/easy-coding.png');
// just display on the fly
$thumb->display(150, 150);
```

== Demo ==
Die Live Demo findet ihr unter demo.easy-coding.de/php/thumbnail/, herunterladen könnt ihr euch den Code dann unter demo.easy-coding.de/php/thumbnail/download.zip