

# Thumbnails in PHP erstellen

== Thumbnail Klasse ==

## Quellcode

```
1. <?php
2. /**
3.  * Generates a thumbnail of given source file image.
4.  *
5.  * @license GNU Lesser General Public License <http://opensource.org/licenses/lgpl-license.php>
6.  */
7. class Thumbnail {
8.     /**
9.      * path to source file
10.     *
11.     * @var string
12.     */
13.     protected $sourceFile = "";
14.     /**
15.      * maximum image width
16.     *
17.     * @var integer
18.     */
19.     protected $maxWidth = 100;
20.     /**
21.      * maximum image height
22.     *
23.     * @var integer
24.     */
25.     protected $maxHeight = 100;
26.     /**
27.      * true, to show source information in thumbnail
28.     *
29.     * @var boolean
30.     */
31.     protected $appendSourceInfo = false;
32.     /**
33.      * true, to prefer embedded thumbnails
34.     *
35.     * @var boolean
36.     */
37.     protected $useEmbedded = true;
38.     /**
39.      * true, to generate quadratic thumbnails
40.     *
41.     * @var boolean
42.     */
43.     protected $quadratic = false;
44.     /**
45.      * mime type of the thumbnail
46.     *
47.     * @var string
48.     */
49.     protected $mimeType = "";
50.     /**
51.      * name of the source image
52.     *
53.     * @var string
54.     */
55.     protected $sourceName = "";
56.     /**
57.      * name of the source image
58.     *
59.     * @var string
60.     */
```

```

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

```

```

127. /**
128. * Creates a thumbnail picture (jpg/png) of a big image
129. *
130. * @param boolean $rescale
131. * @return string thumbnail
132. */
133. protected function makeThumbnail($rescale = false) {
134.     list($width, $height, $this->imageType) = @getImageSize($this->sourceFile);
135.     // check image size
136.     if ($this->checkSize($width, $height, $rescale)) {
137.         return false;
138.     }
139. }
140. // try to extract the embedded thumbnail first (faster)
141. $thumbnail = false;
142. if (!$rescale && $this->useEmbedded) {
143.     $thumbnail = $this->extractEmbeddedThumbnail();
144. }
145. if (!$thumbnail) {
146.     // calculate uncompressed filesize
147.     // and cancel to avoid a memory_limit error
148.     $memoryLimit = ini_get('memory_limit');
149.     if ($memoryLimit != "") {
150.         $memoryLimit = substr($memoryLimit, 0, -1) * 1024 * 1024;
151.         $fileSize = $width * $height * ($this->imageType == 3 ? 4 : 3);
152.         if (($fileSize * 2.1) + memory_get_usage() > ($memoryLimit)) {
153.             return false;
154.         }
155.     }
156. }
157. // calculate new picture size
158. $x = $y = 0;
159. if ($this->quadratic) {
160.     $newWidth = $newHeight = $this->maxWidth;
161.     if ($this->appendSourceInfo) $newHeight -= self::$sourceInfoLineHeight * 2;
162.     if ($width > $height) {
163.         $x = ceil(($width - $height) / 2);
164.         $width = $height;
165.     }
166.     else {
167.         $y = ceil(($height - $width) / 2);
168.         $height = $width;
169.     }
170.     else {
171.         $maxHeight = $this->maxHeight;
172.         if ($this->appendSourceInfo) $maxHeight -= self::$sourceInfoLineHeight * 2;
173.         if ($this->maxWidth / $width < $maxHeight / $height) {
174.             $newWidth = $this->maxWidth;
175.             $newHeight = round($height * ($newWidth / $width));
176.         }
177.         else {
178.             $newHeight = $maxHeight;
179.             $newWidth = round($width * ($newHeight / $height));
180.         }
181.     }
182.     // resize image
183.     $imageResource = false;
184.     // jpeg image
185.     if ($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;
222. // 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. $this->mimeType = 'image/jpeg';
237. }
238. else {
239. return false;
240. }
241. @imageDestroy($imageNew);
243. $thumbnail = ob_get_contents();
244. ob_end_clean();
245. }
246. if ($thumbnail && $this->appendSourceInfo && !$rescale) {
248. $thumbnail = $this->appendSourceInfo($thumbnail);
249. }
250. return $thumbnail;
252. }
253. /**
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.     }
316. }

```

```

332. if (strlen($line2) * $fontWidth < $width) {
333. $textX = intval(round(($width - strlen($line2) * $fontWidth) / 2));
334. }
336. imageString($imageDst, $font, $textX, $height + $textY, $line2, $fontColor);
338. // output image
340. ob_start();
342. if ($this->imageType == 1 && function_exists('imageGIF')) {
343. @imageGIF($imageDst);
344. $this->mimeType = 'image/gif';
345. }
346. else if (($this->imageType == 1 || $this->imageType == 3) && function_exists('imagePNG')) {
347. @imagePNG($imageDst);
348. $this->mimeType = 'image/png';
349. }
350. else if (function_exists('imageJPEG')) {
351. @imageJPEG($imageDst, "", 90);
352. $this->mimeType = 'image/jpeg';
353. }
354. else {
355. return false;
356. }
358. @imageDestroy($imageDst);
359. $thumbnail = ob_get_contents();
360. ob_end_clean();
362. return $thumbnail;
363. }
366. /**
367.  * Extracts the embedded thumbnail picture of a jpeg or tiff image
368.  *
369.  * @return string thumbnail
370.  */
371. protected function extractEmbeddedThumbnail() {
372. if (!function_exists('exif_thumbnail')) {
373. return false;
374. }
376. $width = $height = $type = 0;
377. $thumbnail = @exif_thumbnail($this->sourceFile, $width, $height, $type);
378. if ($thumbnail && $type && $width && $height) {
379. // resize the extracted thumbnail again if necessary
380. // (normally the thumbnail size is set to 160px
381. // which is recommended in EXIF >2.1 and DCF)
382. $this->mimeType = image_type_to_mime_type($type);
383. if (!$this->checkSize($width, $height) && function_exists('sys_get_temp_dir')) {
384. // get temporary file name
385. $this->sourceFile = tempnam(sys_get_temp_dir(), 'Tux');
388. $fh = fopen($this->sourceFile, "w");
389. fwrite($fh, $thumbnail);
390. fclose($fh);
391. unset($thumbnail, $tmpFile);
392. // resize tmp file again
394. return $this->makeThumbnail(true);
395. }
398. return $thumbnail;
399. return false;
401. }
402. /**

```

```

404. * Checks the size of an image.
405. */
406. protected function checkSize($width, $height, $rescale = true) {
407.     $maxHeight = $this->maxHeight;
408.     if ($this->appendSourceInfo && $rescale) {
409.         $maxHeight -= self::$sourceInfoLineHeight * 2;
410.     }
411.     if ($width > $this->maxWidth || $height > $maxHeight) {
412.         return false;
413.     }
414.     return true;
415. }
416. /**
417.  * Returns the mime type of the generated thumbnail.
418.  *
419.  * @return string
420.  */
421. public function getMimeType() {
422.     return $this->mimeType;
423. }
424. /**
425.  * Truncates the given file name to needed length.
426.  *
427.  * @param string $name
428.  * @param string $maxChars
429.  * @return string
430.  */
431. protected static function truncateSourceName($name, $maxChars) {
432.     $extension = "";
433.     $lastPosition = strrpos($name, '.');
434.     if ($lastPosition !== null) {
435.         $extension = substr($name, $lastPosition);
436.         $name = substr($name, 0, $lastPosition);
437.         $maxChars -= strlen($extension);
438.     }
439.     return substr($name, 0, $maxChars - 3) . '...' . $extension;
440. }
441. /**
442.  *
443.  * @return boolean
444.  */
445. public function save($targetFile, $maxWidth = 0, $maxHeight = 0) {
446.     $recover = array();
447.     if ($maxWidth) {
448.         $recover['maxWidth'] = $this->maxWidth;
449.         $this->maxWidth = $maxWidth;
450.     }
451.     if ($maxHeight) {
452.         $recover['maxHeight'] = $this->maxHeight;
453.         $this->maxHeight = $maxHeight;
454.     }
455.     // get thumbnail
456.     try {
457.         $success = true;
458.         if (($thumbnailData = $this->makeThumbnail())) {
459.             // save thumbnail
460.             $fh = fopen($targetFile, 'w');
461.             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) {
479. $this->$key = $val;
480. }
481. return $success;
482. }
483. protected static function sendLastModified($timestamp) {
485. // Getting headers sent by the client.
486. $headers = apache_request_headers();
487. // Checking if the client is validating his cache and if it is current.
489. if (isset($headers['If-Modified-Since']) && (strtotime($headers['If-Modified-Since']) >= $timestamp)) {
490. // this timestamp can also be used to identify the user ;) see
491. // Client's cache IS current, so we just respond '304 Not Modified'.
492. header('Last-Modified: '.gmdate('D, d M Y H:i:s', $timestamp).' GMT', true, 304);
493. exit;
494. } else {
495. // Image not cached or cache outdated, we respond '200 OK' and output the image.
496. header('Last-Modified: '.gmdate('D, d M Y H:i:s', $timestamp).' GMT', true, 200);
497. }
498. }
500. /**
501. * send header and content for the picture
502. */
503. public function display($maxWidth = 100, $maxHeight = 100, $age = '3 MONTH') {
504. $recover = array();
505. if($maxWidth) {
506. $recover['maxWidth'] = $this->maxWidth;
507. $this->maxWidth = $maxWidth;
508. }
509. if($maxHeight) {
510. $recover['maxHeight'] = $this->maxHeight;
511. $this->maxHeight = $maxHeight;
512. }
513. $filemtime = @filemtime($this->sourceFile);
515. $thumbnailData = $this->makeThumbnail();
516. $size = strlen($thumbnailData);
517. self::sendLastModified($filemtime);
519. header('Accept-Ranges: bytes');
520. header('Content-Length: '.$size);
521. header('Content-Type: image/jpeg');
522. header('Expires: '.gmdate('D, d M Y H:i:s', time() + strtotime($age)).' GMT');
523. echo $thumbnailData;
526. foreach($recover as $key => $val) {
527. $this->$key = $val;
528. }
529. }
530. }
531. ?>

```

Alles anzeigen

== Beispiele ==

<https://www.easy-coding.de/wiki/Entry/128-Thumbnails-in-PHP-erstellen/?s=7a0e13a0b48402c57108fcd3e02700c28a9265c1>



=== Bild in mehreren Größen speichern ===

### Quellcode

```
1. <?php
2. require_once('Thumbnail.php');
3. // this is the source
5. $thumb = new Thumbnail('demo/easy-coding.png');
6. // quadratic
8. $thumb->save('/tmp/easy-coding.png', 150, 150);
10. // 4:3 format
11. $thumb->save('/tmp/easy-coding.png', 100, 75);
12. ?>
```

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

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

== Demo ==

Die Live Demo findet ihr unter [demo.easy-coding.de/php/thumbnail/](http://demo.easy-coding.de/php/thumbnail/), herunterladen könnt ihr euch den Code dann unter [demo.easy-coding.de/php/thumbnail/download.zip](http://demo.easy-coding.de/php/thumbnail/download.zip)