easy-thumbnails Documentation

Release 2.1

Chris Beaven

September 02, 2014

Contents

1 Installation **3**
- 1.1 Installing easy-thumbnails . . . 3
- 1.2 Configuring your project . . . 3
- 1.3 Using with Django South . . . 4

2 Usage **5**
- 2.1 Overview . . . 5
- 2.2 Thumbnail aliases . . . 6
- 2.3 Templates . . . 8
- 2.4 Models . . . 10
- 2.5 Python . . . 10

3 Thumbnail Files and Generators **13**

4 Optimizing Images using a Postprocessor **17**
- 4.1 Installation and configuration . . . 17

5 Image Processors **19**
- 5.1 Built-in processors . . . 19
- 5.2 Custom processors . . . 20

6 Settings **23**

7 Source Generators **27**
- 7.1 Built-in generators . . . 27

Python Module Index **29**

This documentation covers the 2.1 release of easy-thumbnails, a thumbnailing application for Django which is easy to use and customize.

To get up and running, consult the *installation guide* which describes all the necessary steps to install and configure this application.

CHAPTER 1

Installation

Before installing easy-thumbnails, you'll obviously need to have copy of Django installed. For the 2.1 release, Django 1.4 and up is supported.

By default, all of the image manipulation is handled by the Python Imaging Library (a.k.a. PIL), so you'll probably want that installed too.

1.1 Installing easy-thumbnails

The easiest way is to use an automatic package-installation tools like pip.

Simply type:

```
pip install easy-thumbnails
```

1.1.1 Manual installation

If you prefer not to use an automated package installer, you can download a copy of easy-thumbnails and install it manually. The latest release package can be downloaded from easy-thumbnail's listing on the Python Package Index.

Once you've downloaded the package, unpack it and run the `setup.py` installation script:

```
python setup.py install
```

1.2 Configuring your project

In your Django project's settings module, add easy-thumbnails to your `INSTALLED_APPS` setting:

```
INSTALLED_APPS = (
    ...
    'easy_thumbnails',
)
```

If you are using Django 1.7, run `python manage.py migrate easy_thumbnails`. Otherwise, just run `python manage.py syncdb`.

You're done! You'll want to head on over now to the *usage documentation*.

1.3 Using with Django South

Django South migrations are stored in the `south_migrations` sub-package.

In order to use South with easy_thumbnails, you will need to customize the `SOUTH_MIGRATION_MODULES` setting:

```
SOUTH_MIGRATION_MODULES = {
    'easy_thumbnails': 'easy_thumbnails.south_migrations',
}
```

CHAPTER 2

Usage

The common way easy-thumbnails is used is via the `{% thumbnail %}` template tag or `thumbnail_url` filter, which generates images from a model with an `ImageField`. The tag can also be used with media images not tied to a specific model by passing in the relative path instead.

Custom database fields are also available for simpler access.

The underlying Python code can be used for lower-level generation of thumbnail images.

2.1 Overview

The primary function of easy-thumbnails is to dynamically create thumbnails based on a source image.

So whenever a thumbnail does not exist or if the source was modified more recently than the existing thumbnail, a new thumbnail is generated (and saved).

Thumbnail aliases can be defined in the `THUMBNAIL_ALIASES` setting, providing predefined thumbnail options. This also allows for generation of thumbnails when the source image is uploaded.

2.1.1 Thumbnail options

To generate a thumbnail of a source image, you specify options which are used by the image processors to generate the required image.

`size` is a required option, and defines the bounds that the generated image must fit within.

Other options are only provided if the given functionality is required:

- `quality=<N>` where N is an integer between 1 and 100 specifying output JPEG quality. The default is 85.
- **`subsampling=<N>` sets the JPEG color subsampling level where N is:**
  - `2` is 4:1:1 (both easy-thumbnails and PIL's default)
  - `1` is 4:2:2 (slightly crisper color borders, small increase in size)
  - `0` is 4:4:4 (very crisp color borders, ~15% increase in size)
- `autocrop` removes any unnecessary whitespace from the edges of the source image.
- `bw` converts the image to grayscale.
- `replace_alpha=#colorcode` replaces any transparency layer with a solid color.
- `crop=<smart|scale|W,H>` cuts the edges of the image to match the aspect ratio of `size` before resizing.

  - *smart* means the image is incrementally cropped down to the requested size by removing slices from edges with the least entropy.
  - *scale* means at least one dimension fits within the size dimensions given.
  - *W,H* modifies the cropping origin behavior:
    * `crop="0,0"` will crop from the left and top edges.
    * `crop="-10,-0"` will crop from the right edge (with a 10% offset) and the bottom edge.
    * `crop=",0"` will keep the default behavior for the x axis (horizontally centering the image) and crop from the top edge.

For a complete and detailed list of options, see the *Image Processors* reference documentation.

To change a default options level, add it to the `THUMBNAIL_DEFAULT_OPTIONS` setting. Be aware that this will change the filename for thumbnails, so existing thumbnails which don't explicitly specify the new default option will have a new filename (and therefore be regenerated).

2.2 Thumbnail aliases

An alias is a specific set of thumbnail options.

Using aliases gives you a single location to define all of your standard thumbnail sizes/options, and avoids repetition of thumbnail options in your code and templates.

An alias may be available project-wide, or set to target a specific app, model or field.

The setting is defined like this:

```
THUMBNAIL_ALIASES = {
    <target>: {
        <alias name>: <alias options dictionary>,
        ...
    },
    ...
}
```

Use the target `''` for project-wide aliases. Otherwise, the target should be a string which defines the scope of the contained aliases:

- `'sprocket.Widget.image'` would make the aliases available to only the 'image' field of a 'Widget' model in an app named 'sprocket'.
- `'sprocket.Widget'` would apply to any field in the 'Widget' model.
- `'sprocket'` would target any field in any model in the app.

2.2.1 Pregeneration

Some provided signal handlers (along with a new `saved_file` signal) allow for you to have the relevant aliases generated when a file is uploaded.

`easy_thumbnails.signal_handlers.`**`generate_aliases`**(*fieldfile*, ***kwargs*)
A saved_file signal handler which generates thumbnails for all field, model, and app specific aliases matching the saved file's field.

easy_thumbnails.signal_handlers.**generate_aliases_global**(*fieldfile*, ***kwargs*)
A saved_file signal handler which generates thumbnails for all field, model, and app specific aliases matching the saved file's field, also generating thumbnails for each project-wide alias.

In a module that will be executed when Django loads (such as a `models.py` file), register one of these signal handlers. For example:

```
from easy_thumbnails.signals import saved_file
from easy_thumbnails.signal_handlers import generate_aliases_global

saved_file.connect(generate_aliases_global)
```

2.2.2 Asynchronous Pregeneration

For some use cases, it may not be necessary to have the relevant aliases generated at the exact moment a file is uploaded. As an alternative, the pregeneration task can be queued and executed by a background process.

The following example uses django-celery in conjunction with Celery to achieve this.

models.py:

```
from django.dispatch import receiver
from easy_thumbnails.signals import saved_file
from myapp import tasks

@receiver(saved_file)
def generate_thumbnails_async(sender, fieldfile, **kwargs):
    tasks.generate_thumbnails.delay(
        model=sender, pk=fieldfile.instance.pk,
        field=fieldfile.field.name)
```

tasks.py:

```
from celery import task
from easy_thumbnails.files import generate_all_aliases

@task
def generate_thumbnails(model, pk, field):
    instance = model._default_manager.get(pk=pk)
    fieldfile = getattr(instance, field)
    generate_all_aliases(fieldfile, include_global=True)
```

This results in a more responsive experience for the user, particularly when dealing with large files and/or remote storage.

2.2.3 Setting aliases for your third-party app

If you have a distributable app that uses easy-thumbnails and want to provide an alias, you can modify the aliases at runtime.

For example, put something like this in a module that will execute when Django initializes (such as `models.py`):

```
from easy_thumbnails.alias import aliases
if not aliases.get('badge'):
    aliases.set('badge', {'size': (150, 80), 'crop': True})
```

2.3 Templates

To make the easy-thumbnail template library available for use in your template, use:

```
{% load thumbnail %}
```

2.3.1 `thumbnail_url` filter

easy_thumbnails.templatetags.thumbnail.**thumbnail_url**(*source*, *alias*)

Return the thumbnail url for a source file using an aliased set of thumbnail options.

If no matching alias is found, returns an empty string.

Example usage:

```
<img src="{{ person.photo|thumbnail_url:'small' }}" alt="">
```

2.3.2 `{% thumbnail %}` tag

If you want to create a thumbnail *without* providing an alias, use this tag to generate the thumbnail by specifying all of the required options (or *with* an alias name, to override/supplement the default alias options with dynamic content).

easy_thumbnails.templatetags.thumbnail.**thumbnail**(*parser*, *token*)

Creates a thumbnail of an ImageField.

Basic tag Syntax:

```
{% thumbnail [source] [size] [options] %}
```

source must be a `File` object, usually an Image/FileField of a model instance.

size can either be:

- the name of an alias
- the size in the format `[width]x[height]` (for example, `{% thumbnail person.photo 100x50 %}`) or
- a variable containing a valid size (i.e. either a string in the `[width]x[height]` format or a tuple containing two integers): `{% thumbnail person.photo size_var %}`.

options are a space separated list of options which are used when processing the image to a thumbnail such as `sharpen`, `crop` and `quality=90`.

If *size* is specified as an alias name, *options* are used to override and/or supplement the options defined in that alias.

The thumbnail tag can also place a `ThumbnailFile` object in the context, providing access to the properties of the thumbnail such as the height and width:

```
{% thumbnail [source] [size] [options] as [variable] %}
```

When `as [variable]` is used, the tag doesn't output anything. Instead, use the variable like a standard `ImageFieldFile` object:

```
{% thumbnail obj.picture 200x200 upscale as thumb %}
<img href="{{ thumb.url }}"
     width="{{ thumb.width }}"
     height="{{ thumb.height }}" />
```

Debugging

By default, if there is an error creating the thumbnail or resolving the image variable then the thumbnail tag will just return an empty string (and if there was a context variable to be set then it will also be set to an empty string).

For example, you will not see an error if the thumbnail could not be written to directory because of permissions error. To display those errors rather than failing silently, set `THUMBNAIL_DEBUG = True` in your Django project's settings module.

For a full list of options, read the *Image Processors* reference documentation.

2.3.3 Fallback images

If you need to support fallback or default images at template level you can use:

```
{% thumbnail object.image|default:'img/default_image.png' 50x50 %}
```

Where the image string is relative to your default storage (usually the `MEDIA_ROOT` setting).

2.3.4 `thumbnailer` filters

There are two filters that you can use if you want to get direct access to a thumbnailer in your template. This can be useful when dealing with aliased thumbnails.

`easy_thumbnails.templatetags.thumbnail.`**`thumbnailer`**(*obj*, *relative_name=None*)

Creates a thumbnailer from an object (usually a `FileField`).

Example usage:

```
{% with photo=person.photo|thumbnailer %}
{% if photo %}
    <a href="{{ photo.large.url }}">
        {{ photo.square.tag }}
    </a>
{% else %}
    <img src="{% static 'template/fallback.png' %}" alt="" />
{% endif %}
{% endwith %}
```

If you know what you're doing, you can also pass the relative name:

```
{% with photo=storage|thumbnailer:'some/file.jpg' %}...
```

`easy_thumbnails.templatetags.thumbnail.`**`thumbnailer_passive`**(*obj*)

Creates a thumbnailer from an object (usually a `FileFile`) that won't generate new thumbnails.

This is useful if you are using another process to generate the thumbnails rather than having them generated on the fly if they are missing.

Example usage:

```
{% with avatar=person.avatar|thumbnailer_passive %}
    {% with avatar_thumb=avatar.small %}
        {% if avatar_thumb %}
            <img src="{{ avatar_thumb.url }}" alt="" />
        {% else %}
            <img src="{% static 'img/default-avatar-small.png' %}"
```

```
                    alt="" />
            {% endif %}
        {% endwith %}
    {% endwith %}
```

2.4 Models

You can use the `ThumbnailerField` or `ThumbnailerImageField` fields (based on `FileField` and `ImageField`, respectively) for easier access to retrieve (or generate) thumbnail images, use different storages and resize source images before saving.

class `easy_thumbnails.fields.`**`ThumbnailerField`**(**args*, ***kwargs*)
A file field which provides easier access for retrieving (and generating) thumbnails.

To use a different file storage for thumbnails, provide the `thumbnail_storage` keyword argument.

class `easy_thumbnails.fields.`**`ThumbnailerImageField`**(**args*, ***kwargs*)
An image field which provides easier access for retrieving (and generating) thumbnails.

To use a different file storage for thumbnails, provide the `thumbnail_storage` keyword argument.

To thumbnail the original source image before saving, provide the `resize_source` keyword argument, passing it a usual thumbnail option dictionary. For example:

```
ThumbnailerImageField(
    ..., resize_source=dict(size=(100, 100), sharpen=True))
```

2.5 Python

Easy thumbnails uses a Django `File`-like object called a `Thumbnailer` to generate thumbnail images from the source file which it references.

2.5.1 `get_thumbnailer`

The easy way to create a `Thumbnailer` instance is to use the following utility function:

`easy_thumbnails.files.`**`get_thumbnailer`**(*obj*, *relative_name=None*)
Get a `Thumbnailer` for a source file.

The `obj` argument is usually either one of the following:

- `FieldFile` instance (i.e. a model instance file/image field property).
- A string, which will be used as the relative name (the source will be set to the default storage).
- `Storage` instance - the `relative_name` argument must also be provided.

Or it could be:

- A file-like instance - the `relative_name` argument must also be provided.

  In this case, the thumbnailer won't use or create a cached reference to the thumbnail (i.e. a new thumbnail will be created for every `Thumbnailer.get_thumbnail()` call).

If `obj` is a `Thumbnailer` instance, it will just be returned. If it's an object with an `easy_thumbnails_thumbnailer` then the attribute is simply returned under the assumption it is a Thumbnailer instance)

Once you have an instance, you can use the `Thumbnailer.get_thumbnail()` method to retrieve a thumbnail, which will (by default) generate it if it doesn't exist (or if the source image has been modified since it was created).

For example, assuming an `aardvark.jpg` image exists in the default storage:

```
from easy_thumbnails.files import get_thumbnailer

thumbnailer = get_thumbnailer('animals/aardvark.jpg')

thumbnail_options = {'crop': True}
for size in (50, 100, 250):
    thumbnail_options.update({'size': (size, size)})
    thumbnailer.get_thumbnail(thumbnail_options)
```

2.5.2 Non-Django file objects

If you need to process a standard file-like object, use `get_thumbnailer()` and provide a `relative_name` like this:

```
picture = open('/home/zookeeper/pictures/my_anteater.jpg')
thumbnailer = get_thumbnailer(picture, relative_name='animals/anteater.jpg')
thumb = thumbnailer.get_thumbnail({'size': (100, 100)})
```

If you don't even need to save the thumbnail to storage because you are planning on using it in some more direct way, you can use the `Thumbnailer.generate_thumbnail()` method.

Thumbnails generated in this manor don't use any cache reference, i.e. every call to `Thumbnailer.get_thumbnail()` will generate a fresh thumbnail image.

Reference documentation:

CHAPTER 3

Thumbnail Files and Generators

Following is some basic documentation of the classes and methods related to thumbnail files and lower level generation.

class easy_thumbnails.files.**ThumbnailFile**(*name*, *file=None*, *storage=None*, *thumbnail_options=None*, **args*, ***kwargs*)

A thumbnailed file.

This can be used just like a Django model instance's property for a file field (i.e. an ImageFieldFile object).

image

Get a PIL Image instance of this file.

The image is cached to avoid the file needing to be read again if the function is called again.

set_image_dimensions(*thumbnail*)

Set image dimensions from the cached dimensions of a Thumbnail model instance.

tag(*alt=''*, *use_size=None*, ***attrs*)

Return a standard XHTML tag for this field.

Parameters

- **alt** – The alt="" text for the tag. Defaults to ''.
- **use_size** – Whether to get the size of the thumbnail image for use in the tag attributes. If None (default), the size will only be used it if won't result in a remote file retrieval.

All other keyword parameters are added as (properly escaped) extra attributes to the *img* tag.

class easy_thumbnails.files.**Thumbnailer**(*file=None*, *name=None*, *source_storage=None*, *thumbnail_storage=None*, *remote_source=False*, *generate=True*, **args*, ***kwargs*)

A file-like object which provides some methods to generate thumbnail images.

You can subclass this object and override the following properties to change the defaults (pulled from the default settings):

•source_generators

•thumbnail_processors

generate_thumbnail(*thumbnail_options*, *high_resolution=False*, *silent_template_exception=False*)

Return an unsaved ThumbnailFile containing a thumbnail image.

The thumbnail image is generated using the thumbnail_options dictionary.

get_existing_thumbnail(*thumbnail_options*, *high_resolution=False*)
Return a `ThumbnailFile` containing an existing thumbnail for a set of thumbnail options, or `None` if not found.

get_options(*thumbnail_options*, ***kwargs*)
Get the thumbnail options that includes the default options for this thumbnailer (and the project-wide default options).

get_thumbnail(*thumbnail_options*, *save=True*, *generate=None*, *silent_template_exception=False*)
Return a `ThumbnailFile` containing a thumbnail.

If a matching thumbnail already exists, it will simply be returned.

By default (unless the `Thumbnailer` was instanciated with `generate=False`), thumbnails that don't exist are generated. Otherwise `None` is returned.

Force the generation behaviour by setting the `generate` param to either `True` or `False` as required.

The new thumbnail image is generated using the `thumbnail_options` dictionary. If the `save` argument is `True` (default), the generated thumbnail will be saved too.

get_thumbnail_name(*thumbnail_options*, *transparent=False*, *high_resolution=False*)
Return a thumbnail filename for the given `thumbnail_options` dictionary and `source_name` (which defaults to the File's `name` if not provided).

save_thumbnail(*thumbnail*)
Save a thumbnail to the thumbnail_storage.

Also triggers the `thumbnail_created` signal and caches the thumbnail values and dimensions for future lookups.

source_generators = None
A list of source generators to use. If `None`, will use the default generators defined in settings.

thumbnail_exists(*thumbnail_name*)
Calculate whether the thumbnail already exists and that the source is not newer than the thumbnail.

If the source and thumbnail file storages are local, their file modification times are used. Otherwise the database cached modification times are used.

thumbnail_processors = None
A list of thumbnail processors. If `None`, will use the default processors defined in settings.

class `easy_thumbnails.files.`**ThumbnailerFieldFile**(**args*, ***kwargs*)
A field file which provides some methods for generating (and returning) thumbnail images.

delete(**args*, ***kwargs*)
Delete the image, along with any generated thumbnails.

delete_thumbnails(*source_cache=None*)
Delete any thumbnails generated from the source image.

Parameters **source_cache** – An optional argument only used for optimisation where the source cache instance is already known.

Returns The number of files deleted.

get_thumbnails(**args*, ***kwargs*)
Return an iterator which returns ThumbnailFile instances.

save(*name*, *content*, **args*, ***kwargs*)
Save the file, also saving a reference to the thumbnail cache Source model.

class easy_thumbnails.files.**ThumbnailerImageFieldFile**(**args*, ***kwargs*)
A field file which provides some methods for generating (and returning) thumbnail images.

save(*name*, *content*, **args*, ***kwargs*)
Save the image.

The image will be resized down using a ThumbnailField if resize_source (a dictionary of thumbnail options) is provided by the field.

easy_thumbnails.files.**database_get_image_dimensions**(*file*, *close=False*, *dimensions=None*)
Returns the (width, height) of an image, given ThumbnailFile. Set 'close' to True to close the file at the end if it is initially in an open state.

Will attempt to get the dimensions from the file itself if they aren't in the db.

easy_thumbnails.files.**generate_all_aliases**(*fieldfile*, *include_global*)
Generate all of a file's aliases.

Parameters

- **fieldfile** – A FieldFile instance.
- **include_global** – A boolean which determines whether to generate thumbnails for project-wide aliases in addition to field, model, and app specific aliases.

easy_thumbnails.files.**get_thumbnailer**(*obj*, *relative_name=None*)
Get a Thumbnailer for a source file.

The obj argument is usually either one of the following:

- FieldFile instance (i.e. a model instance file/image field property).
- A string, which will be used as the relative name (the source will be set to the default storage).
- Storage instance - the relative_name argument must also be provided.

Or it could be:

- A file-like instance - the relative_name argument must also be provided.

  In this case, the thumbnailer won't use or create a cached reference to the thumbnail (i.e. a new thumbnail will be created for every Thumbnailer.get_thumbnail() call).

If obj is a Thumbnailer instance, it will just be returned. If it's an object with an easy_thumbnails_thumbnailer then the attribute is simply returned under the assumption it is a Thumbnailer instance)

CHAPTER 4

Optimizing Images using a Postprocessor

The PIL and the Pillow libraries do a great job when is comes to crop, rotate or resize images. However, they both operate poorly when it comes to optimizing the payload of the generated files.

For this feature, two portable command line programs can fill the gap: jpegoptim and optipng. They both are open source, run on a huge range of platforms, and can reduce the file size of an image by often more than 50% without loss of quality.

Optimizing such images is a big benefit in terms of loading time and is therefore strongly recommended by tools such as Google's PageSpeed. Moreover, if every website operator cares about, it reduces the overall Internet traffic and thus greenhouse gases by some googolth percent.

Support for these postprocessors (or other similar ones) is available as an optional feature in easy-thumbnails.

4.1 Installation and configuration

Install one or both of the above programs on your operating system.

In your Django project's settings module, add the optimizing postprocessor to your configuration settings:

```
INSTALLED_APP = (
    ...
    'easy_thumbnails',
    'easy_thumbnails.optimize',
    ...
)
```

There is one configuration settings dictionary:

`OptimizeSettings.`**`THUMBNAIL_OPTIMIZE_COMMAND`** **= {'gif': None, 'jpeg': None, 'png': None}**

Postprocess thumbnails of type PNG, GIF or JEPG after transformation but before storage.

Apply an external post processing program to images after they have been manipulated by PIL or Pillow. This is strongly recommended by tools such as Google's PageSpeed on order to reduce the payload of the thumbnailed image files.

Example:

```
THUMBNAIL_OPTIMIZE_COMMAND = {
    'png': '/usr/bin/optipng {filename}',
    'gif': '/usr/bin/optipng {filename}',
    'jpeg': '/usr/bin/jpegoptim {filename}'
}
```

Note that `optipng` can also optimize images of type GIF.

CHAPTER 5

Image Processors

Easy thumbnails generates thumbnail images by passing the source image through a series of image processors. Each processor may alter the image, often dependent on the options it receives.

This makes the system very flexible, as the processors an image passes through can be defined in `THUMBNAIL_PROCESSORS` and even overridden by an individual `easy_thumbnails.files.Thumbnailer` (via the `thumbnail_processors` attribute).

5.1 Built-in processors

Following is a list of the built-in processors, along with the thumbnail options which they use.

`easy_thumbnails.processors.`**`autocrop`**(*im*, *autocrop=False*, ***kwargs*)
Remove any unnecessary whitespace from the edges of the source image.

This processor should be listed before `scale_and_crop()` so the whitespace is removed from the source image before it is resized.

autocrop Activates the autocrop method for this image.

`easy_thumbnails.processors.`**`background`**(*im*, *size*, *background=None*, *crop=None*, *replace_alpha=None*, ***kwargs*)
Add borders of a certain color to make the resized image fit exactly within the dimensions given.

background Background color to use

`easy_thumbnails.processors.`**`colorspace`**(*im*, *bw=False*, *replace_alpha=False*, ***kwargs*)
Convert images to the correct color space.

A passive option (i.e. always processed) of this method is that all images (unless grayscale) are converted to RGB colorspace.

This processor should be listed before `scale_and_crop()` so palette is changed before the image is resized.

bw Make the thumbnail grayscale (not really just black & white).

replace_alpha Replace any transparency layer with a solid color. For example, `replace_alpha='#fff'` would replace the transparency layer with white.

`easy_thumbnails.processors.`**`filters`**(*im*, *detail=False*, *sharpen=False*, ***kwargs*)
Pass the source image through post-processing filters.

sharpen Sharpen the thumbnail image (using the PIL sharpen filter)

detail Add detail to the image, like a mild *sharpen* (using the PIL `detail` filter).

easy_thumbnails.processors.**scale_and_crop**(*im*, *size*, *crop=False*, *upscale=False*, *zoom=None*, *target=None*, ***kwargs*)

Handle scaling and cropping the source image.

Images can be scaled / cropped against a single dimension by using zero as the placeholder in the size. For example, `size=(100, 0)` will cause the image to be resized to 100 pixels wide, keeping the aspect ratio of the source image.

crop Crop the source image height or width to exactly match the requested thumbnail size (the default is to proportionally resize the source image to fit within the requested thumbnail size).

By default, the image is centered before being cropped. To crop from the edges, pass a comma separated string containing the `x` and `y` percentage offsets (negative values go from the right/bottom). Some examples follow:

- `crop="0,0"` will crop from the left and top edges.
- `crop="-10,-0"` will crop from the right edge (with a 10% offset) and the bottom edge.
- `crop=",0"` will keep the default behavior for the x axis (horizontally centering the image) and crop from the top edge.

The image can also be "smart cropped" by using `crop="smart"`. The image is incrementally cropped down to the requested size by removing slices from edges with the least entropy.

Finally, you can use `crop="scale"` to simply scale the image so that at least one dimension fits within the size dimensions given (you may want to use the upscale option too).

upscale Allow upscaling of the source image during scaling.

zoom A percentage to zoom in on the scaled image. For example, a zoom of `40` will clip 20% off each side of the source image before thumbnailing.

target Set the focal point as a percentage for the image if it needs to be cropped (defaults to `(50, 50)`).

For example, `target="10,20"` will set the focal point as 10% and 20% from the left and top of the image, respectively. If the image needs to be cropped, it will trim off the right and bottom edges until the focal point is centered.

Can either be set as a two-item tuple such as `(20, 30)` or a comma separated string such as `"20,10"`.

A null value such as `(20, None)` or `",60"` will default to 50%.

5.2 Custom processors

You can replace or leave out any default processor as suits your needs. Following is an explanation of how to create and activate a custom processor.

When defining the `THUMBNAIL_PROCESSORS` setting, remember that this is the order through which the processors are run. The image received by a processor is the output of the previous processor.

5.2.1 Create the processor

First create a processor like this:

```
def whizzbang_processor(image, bang=False, **kwargs):
    """
    Whizz bang the source image.

    """
```

```
    if bang:
        image = whizz(image)
    return image
```

The first argument for a processor is the source image.

All other arguments are keyword arguments which relate to the list of options received by the thumbnail generator (including `size` and `quality`). Ensure you list all arguments which could be used (giving them a default value of `False`), as the processors arguments are introspected to generate a list of valid options.

You must also use `**kwargs` at the end of your argument list because all options used to generate the thumbnail are passed to processors, not just the ones defined.

Whether a processor actually modifies the image or not, they must always return an image.

5.2.2 Use the processor

Next, add the processor to `THUMBNAIL_PROCESSORS` in your settings module:

```
from easy_thumbnails.conf import Settings as easy_thumbnails_defaults

THUMBNAIL_PROCESSORS = easy_thumbnails_defaults.THUMBNAIL_PROCESSORS + (
    'wb_project.thumbnail_processors.whizzbang_processor',
)
```

CHAPTER 6

Settings

class easy_thumbnails.conf.**Settings**

These default settings for easy-thumbnails can be specified in your Django project's settings module to alter the behaviour of easy-thumbnails.

THUMBNAIL_ALIASES = None

A dictionary of predefined alias options for different targets. See the *usage documentation* for details.

THUMBNAIL_BASEDIR = ''

Save thumbnail images to a directory directly off MEDIA_ROOT, still keeping the relative directory structure of the source image.

For example, using the {% thumbnail "photos/1.jpg" 150x150 %} tag with a THUMBNAIL_BASEDIR of 'thumbs' would result in the following thumbnail filename:

```
MEDIA_ROOT + 'thumbs/photos/1_jpg_150x150_q85.jpg'
```

THUMBNAIL_CACHE_DIMENSIONS = False

Save thumbnail dimensions to the database.

When using remote storage backends it can be a slow process to get image dimensions for a thumbnailed file. This option will store them in the database to be recalled quickly when required. Note: the old method still works as a fall back.

THUMBNAIL_CHECK_CACHE_MISS = False

If this boolean setting is set to True, and a thumbnail cannot be found in the database tables, we ask the storage if it has the thumbnail. If it does we add the row in the database, and we don't need to generate the thumbnail.

Switch this to True if your easy_thumbnails_thumbnail table has been wiped but your storage still has the thumbnail files.

THUMBNAIL_DEBUG = False

If this boolean setting is set to True, display errors creating a thumbnail when using the *{% thumbnail %} tag* rather than failing silently.

THUMBNAIL_DEFAULT_OPTIONS = None

Set this to a dictionary of options to provide as the default for all thumbnail calls. For example, to make all images greyscale:

```
THUMBNAIL_DEFAULT_OPTIONS = {'bw': True}
```

THUMBNAIL_DEFAULT_STORAGE = 'easy_thumbnails.storage.ThumbnailFileSystemStorage'

The default Django storage for *saving* generated thumbnails.

THUMBNAIL_EXTENSION = 'jpg'
The type of image to save thumbnails with no transparency layer as.

Note that changing the extension will most likely cause the `THUMBNAIL_QUALITY` setting to have no effect.

THUMBNAIL_HIGHRES_INFIX = '@2x'
Sets the infix used to distinguish thumbnail images for retina displays.

Thumbnails generated for retina displays are distinguished from the standard resolution counterparts, by adding an infix to the filename just before the dot followed by the extension.

Apple Inc., formerly suggested to use `@2x` as infix, but later changed their mind and now suggests to use `_2x`, since this is more portable.

THUMBNAIL_HIGH_RESOLUTION = False
Enables thumbnails for retina displays.

Creates a version of the thumbnails in high resolution that can be used by a javascript layer to display higher quality thumbnails for high DPI displays.

This can be overridden at a per-thumbnail level with the `HIGH_RESOLUTION` thumbnail option:

```
opts = {'size': (100, 100), 'crop': True, HIGH_RESOLUTION: False}
only_basic = get_thumbnailer(obj.image).get_thumbnail(opts)
```

In a template tag, use a value of `0` to force the disabling of a high resolution version or just the option name to enable it:

```
{% thumbnail obj.image 50x50 crop HIGH_RESOLUTION=0 %}  {# no hires #}
{% thumbnail obj.image 50x50 crop HIGH_RESOLUTION %}  {# force hires #}
```

THUMBNAIL_MEDIA_ROOT = ''
Used by easy-thumbnail's default storage to locate where thumbnails are stored on the file system.

If not provided, Django's standard `MEDIA_ROOT` setting is used.

THUMBNAIL_MEDIA_URL = ''
Used by easy-thumbnail's default storage to build the absolute URL for a generated thumbnail.

If not provided, Django's standard `MEDIA_URL` setting is used.

THUMBNAIL_NAMER = 'easy_thumbnails.namers.default'
The function used to generate the filename for thumbnail images.

Three namers are included in easy_thumbnails:

easy_thumbnails.namers.default Descriptive filename containing the source and options like `source.jpg.100x100_q80_crop_upscale.jpg`.

easy_thumbnails.namers.hashed Short hashed filename like `1xedFtqllFo9.jpg`.

easy_thumbnails.namers.source_hashed Filename with source hashed, size, then options hashed like `1xedFtqllFo9_100x100_QHCa6G1l.jpg`.

To write a custom namer, always catch all other keyword arguments arguments (with **kwargs). You have access to the following arguments: `thumbnailer`, `source_filename`, `thumbnail_extension` (does *not* include the `'.'`), `thumbnail_options`, `prepared_options`.

The `thumbnail_options` are a copy of the options dictionary used to build the thumbnail, `prepared_options` is a list of options prepared as text, and excluding options that shouldn't be included in the filename.

THUMBNAIL_PREFIX = ‘’

Prepend thumbnail filenames with the specified prefix.

For example, using the `{% thumbnail "photos/1.jpg" 150x150 %}` tag with a `THUMBNAIL_PREFIX` of `'thumbs_'` would result in the following thumbnail filename:

```
MEDIA_ROOT + 'photos/thumbs_1_jpg_150x150_q85.jpg'
```

THUMBNAIL_PRESERVE_EXTENSIONS = None

To preserve specific extensions, for instance if you always want to create lossless PNG thumbnails from PNG sources, you can specify these extensions using this setting, for example:

```
THUMBNAIL_PRESERVE_EXTENSIONS = ('png',)
```

All extensions should be lowercase.

Instead of a tuple, you can also set this to `True` in order to always preserve the original extension.

THUMBNAIL_PROCESSORS = (‘easy_thumbnails.processors.colorspace’, ‘easy_thumbnails.processors.autocrop’, ‘easy_th

Defaults to:

```
THUMBNAIL_PROCESSORS = (
    'easy_thumbnails.processors.colorspace',
    'easy_thumbnails.processors.autocrop',
    'easy_thumbnails.processors.scale_and_crop',
    'easy_thumbnails.processors.filters',
    'easy_thumbnails.processors.background',
)
```

The *Image Processors* through which the source image is run when you create a thumbnail.

The order of the processors is the order in which they are sequentially called to process the image.

THUMBNAIL_PROGRESSIVE = 100

Use progressive JPGs for thumbnails where either dimension is at least this many pixels.

For example, a 90x90 image will be saved as a baseline JPG while a 728x90 image will be saved as a progressive JPG.

Set to `False` to never use progressive encoding.

THUMBNAIL_QUALITY = 85

The default quality level for JPG images on a scale from 1 (worst) to 95 (best). Technically, values up to 100 are allowed, but this is not recommended.

THUMBNAIL_SOURCE_GENERATORS = (‘easy_thumbnails.source_generators.pil_image’,)

The *Source Generators* through which the base image is created from the source file.

The order of the processors is the order in which they are sequentially tried.

THUMBNAIL_SUBDIR = ‘’

Save thumbnail images to a sub-directory relative to the source image.

For example, using the `{% thumbnail "photos/1.jpg" 150x150 %}` tag with a `THUMBNAIL_SUBDIR` of `'thumbs'` would result in the following thumbnail filename:

```
MEDIA_ROOT + 'photos/thumbs/1_jpg_150x150_q85.jpg'
```

THUMBNAIL_TRANSPARENCY_EXTENSION = ‘png’

The type of image to save thumbnails with a transparency layer (e.g. GIFs or transparent PNGs).

CHAPTER 7

Source Generators

easy-thumbnails allows you to add to (or completely replace) the way that the base image is generated from the source file.

An example of a custom source generator would be one that generated a source image from a video file which could then be used to generate the appropriate thumbnail.

7.1 Built-in generators

easy_thumbnails.source_generators.**pil_image**(*source*, *exif_orientation=True*, ***options*)

Try to open the source file directly using PIL, ignoring any errors.

exif_orientation

If EXIF orientation data is present, perform any required reorientation before passing the data along the processing pipeline.

e

easy_thumbnails.files, 13
easy_thumbnails.processors, 19
easy_thumbnails.signal_handlers, 6
easy_thumbnails.source_generators, 27