python-thumbnails Documentation

Release 1.0.0

Rolf Erik Lekang

April 19, 2015

Contents

Thumbnails for Django, Flask and other Python projects.

CHAPTER 1

Quickstart

1.1 Installation

```
pip install pillow  # default image engine, not necessary if another engine is used
pip install python-thumbnails
```

1.1.1 Dependencies

This project has configurable parts that depends on other modules. In order to use those the dependencies need to be installed, e.g. to use the `PillowEngine` which is the default image engine one has to install pillow.

1.2 Usage

Using python-thumbnails can be as little effort as calling `get_thumbnail`. It works without configuration, even in Django projects.

```
from thumbnails import get_thumbnail

get_thumbnail('path/to/image.png', '300x300', crop='center')
```

1.3 Configuration

It is possible to put settings in a Python module and specify it with the environment variable `THUMBNAILS_SETTINGS_MODULE`.

1.3.1 Django projects

This project integrates with Django without any specific configuration, put your thumbnails settings within your Django settings and you should be good to go. However, if you want to use the templatetag it is necessary to add `thumbnails` to installed apps:

```
INSTALLED_APPS = (
    # your other apps
    'thumbnails',
)
```

1.3.2 Flask projects

Use `THUMBNAILS_SETTINGS_MODULE` as described above. Better integrations with Flask is planned in feature versions.

Quickstart

Installation

```
pip install pillow  # default image engine, not necessary if another engine is used
pip install python-thumbnails
```

Dependencies This project has configurable parts that depends on other modules. In order to use those the dependencies need to be installed, e.g. to use the `PillowEngine` which is the default image engine one has to install pillow.

Usage

Using python-thumbnails can be as little effort as calling `get_thumbnail`. It works without configuration, even in Django projects.

```
from thumbnails import get_thumbnail

get_thumbnail('path/to/image.png', '300x300', crop='center')
```

Configuration

It is possible to put settings in a Python module and specify it with the environment variable `THUMBNAILS_SETTINGS_MODULE`.

Django projects This project integrates with Django without any specific configuration, put your thumbnails settings within your Django settings and you should be good to go. However, if you want to use the templatetag it is necessary to add `thumbnails` to installed apps:

```
INSTALLED_APPS = (
    # your other apps

    'thumbnails',
)
```

Flask projects Use `THUMBNAILS_SETTINGS_MODULE` as described above. Better integrations with Flask is planned in feature versions.

How python-thumbnail works

python-thumbnails have a single function that should be used for most thumbnail generations. It is `thumbnails.get_thumbnail`. That function will use the correct backends to fetch or create the thumbnail.

The first that will happen when that function is called is that a hash is generated based on the image path or url, the wanted size, the wanted crop and other options that are given that differ from the defaults [1]. That hash is used to lookup in cache and in the file path of the thumbnail.

After the hash is created the thumbnail info will be retrieved from the cache, if it is in the cache the function will return with the info about the thumbnail. In the case that the cache has no information about the given hash the storage system will be checked to se if there is a saved thumbnail for that hash. If there is, info about the thumbnail is returned.

If there exist no thumbnail file for the given hash the thumbnail will be created using the current image engine and saved with the storage engine before saving the information about the created thumbnail in the cache.

After creating the thumbnail an extra thumbnail will be created for every resolution listed in the alternative resolution setting. Note that the there is no check if the alternative resolution exists. The alternative resolution thumbnails will only be created if the standard resolution thumbnail does not exist.

Creating thumbnails

`thumbnails.get_thumbnail` should be used to generate thumbnails from python code. It takes the original image and size is positional arguments and needs to be passed each time the function is called. Other options can be passed as keyword arguments. The available options is listed below:

Django specific features

Templatetags get_thumbnail

This templatetag is a shortcut for `thumbnails.get_thumbnail`, thus all arguments and keyword arguments are the same as described in the section above. It is necessary to define the variable name for the thumbnail with an `as` keyword as shown in the example below.

```
{% load thumbnails %}

{% get_thumbnail "image.jpg" "400x400" crop="center" as thumbnail %}
<img src="{{ thumbnail.url }}" alt="The thumbnail" style="width: {{ thumbnail.width }} />
```

Filters

```
{% load thumbnails %}

{{ content|markdown_thumbnails }}
```

```
{% load thumbnails %}

{{ content|html_thumbnails }}
```

```
{% load thumbnails %}

{{ content|safe_html_thumbnails }}
```

[1] The reason for only adding options that differ from the defaults is to avoid massive regeneration of thumbnails if the defaults changes.

Creating custom text filters

It is possible to create custom text filters by utilizing the `text_filter` function described below.

Below is the code for the `html_thumbnails`-filter shown as an example of how to use `text_filter`.

```
@register.filter
def html_thumbnails(value):
    return mark_safe(text_filter('<img(?: alt="(%(caption)s)?")? src="(%(image)s)"', value))
```

Image engines

python-thumbnails uses interchangeable image engines to make it possible to use the imaging framework you like the best. python-thumbnails should have a few to select from. However, if the framework you want to use is not supported. Extend `BaseThumbnailEngine` to create your own. If you think your engine is valuable to others a pull-request is always appreciated.

Cache backends

The cache backends is used to store references to already created thumbnails in order to avoid unnecessary disk or network usage. The `Thumbnail` object is cached and will be returned directly in `get_thumbnail` if the cache returns it.

Storage backends

Settings

`THUMBNAIL_PATH`

The path where thumbnails are saved.

Default: `os.getcwd() + '/thumbnails-cache'`

Default in Django: `django.conf.settings.MEDIA_ROOT + '/thumbnails-cache'`

`THUMBNAIL_URL`

The prefix for the url property of an Thumbnail object. The url property will contain this prefix and the relative path from `THUMBNAIL_PATH`.

Default: `'/thumbnails/''`

Default in Django: `django.conf.settings.MEDIA_URL + '/thumbnails-cache'`

`THUMBNAIL_ENGINE`

Default: `thumbnails.engines.PillowEngine`

`THUMBNAIL_CACHE_BACKEND`

Default: `thumbnails.cache_backends.SimpleCacheBackend`

Default in Django: `thumbnails.cache_backends.DjangoCacheBackend`

`THUMBNAIL_CACHE_TIMEOUT`

A timeout parameter for cache backends that does not support eternal items.

Default: `60 * 60 * 24 * 365` (a year in seconds)

`THUMBNAIL_STORAGE_BACKEND`

Default: `thumbnails.storage_backends.FilesystemStorageBackend`
Default in Django: `thumbnails.storage_backends.DjangoStorageBackend`

Image options

`THUMBNAIL_SCALE_UP`

If this is set to `True` the thumbnails can be scaled bigger than the original.
Default: `False`

`THUMBNAIL_QUALITY`

Quality sent to the engine.
Default: `90`

`THUMBNAIL_COLORMODE`

The default colormode for thumbnails. Supports all values supported by pillow. In other engines there is a best effort translation from pillow modes to the modes supported by the current engine.
Default: `'RGB'`

`THUMBNAIL_FALLBACK_FORMAT`

If the engine is not able to detect file type from the source or the file type is not supported this format will be used.
Defaults: `'JPEG'`

`THUMBNAIL_FALLBACK_FORMAT`

This will override the original image format, however, passing format into `get_thumbnail` will override this value.
Defaults: `None`

`THUMBNAIL_ALTERNATIVE_RESOLUTIONS`

Defines which alternative resolutions should be created. Each item in the list will create an alternative version with the number as a proportions-factor.
Default: `[2]`

Templatetags and filters

`THUMBNAIL_FILTER_OPTIONS`

The options passed into `get_thumbnail` by the Markdown and HTML filter. It can contain all options that is supported by `get_thumbnails`, however size is required.
Default: `{'size': '500'}`

Dummy thumbnails

`THUMBNAIL_DUMMY`

Activates the dummy thumbnail functionality, when this is active the original image will not be opened.
Default: *False*

`THUMBNAIL_DUMMY_FALLBACK`

Makes the dummy thumbnail functionality only be used if the thumbnail cannot be created.
Default: *False*

`THUMBNAIL_DUMMY_URL`

This is the url that the dummy url is generated from. It should be a string that can be used with `string.format` and the arguments are width and height, `THUMBNAIL_DUMMY_URL.format(width=width, height=height)`
Default: *http://puppies.lkng.me/{width}x{height}*

Changelog

Changes since last release

Nothing yet.

0.5.0

- Drop support for Python 2
- Add django filters for markdown and html
- Tested against release version of Django 1.8
- Add `THUMBNAIL_FORCE_FORMAT`
- Add `THUMBNAIL_FALLBACK_FORMAT`
- Change `THUMBNAIL_DUMMY_URL` to use keyword arguments in string format

0.4.1

- Add missing call of colormode in engine.create

0.4.0

- Add support for base64 encoded images as source
- Add support for colormode through setting the option `colormode` or the setting `THUMBNAIL_COLORMODE`.

0.3.0

- Add Django templatetag `get_thumbnail`
- Catch IOError and OSError in PillowEngine.engine_load_image and throw ThumbnailError, which will be caught in get_thumbnail if `THUMBNAIL_DEBUG = False`.

0.2.1

0.2.0

- PgmagickEngine
- WandEngine
- Higher maxblock in PillowEngine.engine_raw_data

0.1.0

- Expendable image engine, storage backend and cache backend
- PillowBackend
- FilesystemStorageBackend
- SimpleCacheBackend
- RedisCacheBackend
- Django integrations
  - DjangoStorageBackend
  - DjangoCacheBackend
  - Settings integration and defaults for django
- Support for dummy thumbnails

Contributing

Contributions are highly appreciated, please follow the following guidelines in order to make the process of including the contributions easier.

It sums up to write tests, follow pep8 and the import-sorting guidelines. New features needs to be documented.

Please remember to add your change to the CHANGELOG.rst under the section "Changes since last release".

Adding features

Adding an engine If you are adding a new engine there is already a test-case ready that you should use to test your new engine. In `tests.test_engine` there is a EngineTestMixin that you should add to your test case. Remember to set the `ENGINE` attribute. If your engine has their own dependencies it is necessary to decorate the test class with `@unittest.skipIf(not has_dependency(), 'Dependency not installed').` The `has_dependency` function should be created in `test.utils`. There exist some in the code base already so look there for examples.

Below is the test case for the PillowEngine. It is a good example of how to add tests for a new engine.

```
@unittest.skipIf(not has_installed('pillow'), 'Pillow not installed')
class PillowEngineTestCase(EngineTestMixin, unittest.TestCase):
    ENGINE = PillowEngine
```

Adding a cache backend As with the image engines, cache backends has a test mixin that should be used when a new cache backend is created. Example usage of the test mixin is shown below.

```
class SimpleCacheBackendTestCase(CacheBackendTestMixin, unittest.TestCase):
    BACKEND = SimpleCacheBackend
```