sorl-thumbnail Documentation

Release 12.4a1

Mikko Hellsing

May 17, 2017

Contents

Contents:

CHAPTER 1

Examples

Template examples

All of the examples assume that you first load the `thumbnail` template tag in your template:

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

Simple:

```
{% thumbnail item.image "100x100" crop="center" as im %}
    <img src="{{ im.url }}" width="{{ im.width }}" height="{{ im.height }}">
{% endthumbnail %}
```

Crop using margin filter, x, y aliases:

```
{% thumbnail item.image "100x700" as im %}
    <img style="margin:{{ im|margin:"100x700" }}" src="{{ im.url }}" width="{{ im.x }}
↪" height="{{ im.y }}">
{% endthumbnail %}
```

Using external images and advanced cropping:

```
{% thumbnail "http://www.aino.se/media/i/logo.png" "40x40" crop="80% top" as im %}
    <img src="{{ im.url }}">
{% endthumbnail %}
```

Using the empty feature, the empty section is rendered when the source is resolved to an empty value or an invalid image source, you can think of it as rendering when the thumbnail becomes undefined:

```
{% thumbnail item.image my_size_string crop="left" as im %}
    <img src="{{ im.url }}">
{% empty %}
    <p>No image</p>
{% endthumbnail %}
```

Nesting tags and setting size (geometry) for width only:

```
{% thumbnail item.image "1000" as big %}
    {% thumbnail item.image "50x50" crop="center" as small %}
        <a href="{{ big.url}}" title="look ma!"><img src="{{ small.url }}"></a>
    {% endthumbnail %}
{% endthumbnail %}
```

Setting geometry for height only:

```
{% thumbnail item.image "x300" as im %}
    <img src="{{ im.url }}">
{% endthumbnail %}
```

Setting format and using the is_portrait filter:

```
{% if item.image|is_portrait %}
    <div class="portrait">
    {% thumbnail item.image "100" crop="10px 10px" format="PNG" as im %}
        <img src="{{ im.url }}">
    {% endthumbnail %}
    </div>
{% else %}
    <div class="landscape">
    {% thumbnail item.image "50" crop="bottom" format="PNG" as im %}
        <img src="{{ im.url }}">
    {% endthumbnail %}
    </div>
    <div>
        <p>Undefined behaviour</p>
    </div>
{% endif %}
```

Using HTML filter:

```
{{ text|html_thumbnails }}
```

Using markdown filter:

```
{{ text|markdown_thumbnails }}
```

Model examples

Using the ImageField that automatically deletes references to itself in the key value store and its thumbnail references when deleted:

```
from django.db import models
from sorl.thumbnail import ImageField

class Item(models.Model):
    image = ImageField(upload_to='whatever')
```

Note: You do not need to use the `sorl.thumbnail.ImageField` to use `sorl.thumbnail`. The standard `django.db.models.ImageField` is fine except that using the `sorl.thumbnail.ImageField` lets you

plugin the nice admin addition explained in the next section.

Another example on how to use `sorl.thumbnail.ImageField` in your existing project with only small code changes:

```
# util/models.py
from django.db.models import *
from sorl.thumbnail import ImageField

# myapp/models.py
from util import models

class MyModel(models.Model):
    logo = models.ImageField(upload_to='/dev/null')
```

Admin examples

Recommended usage using `sorl.thumbnail.admin.AdminImageMixin` (note that this requires use of `sorl.thumbnail.ImageField` in your models as explained above):

```
# myapp/admin.py
from django.contrib import admin
from myapp.models import MyModel
from sorl.thumbnail.admin import AdminImageMixin

class MyModelAdmin(AdminImageMixin, admin.ModelAdmin):
    pass
```

And the same thing For inlines:

```
# myapp/admin.py
from django.contrib import admin
from myapp.models import MyModel, MyInlineModel
from sorl.thumbnail.admin import AdminImageMixin

class MyInlineModelAdmin(AdminImageMixin, admin.TabularInline):
    model = MyInlineModel

class MyModelAdmin(admin.ModelAdmin):
    inlines = [MyInlineModelAdmin]
```

Easy to plugin solution example with little code to change:

```
# util/admin.py
from django.contrib.admin import *
from sorl.thumbnail.admin import AdminImageMixin

class ModelAdmin(AdminImageMixin, ModelAdmin):
    pass

class TabularInline(AdminImageMixin, TabularInline):
    pass

class StackedInline(AdminImageMixin, StackedInline):
    pass
```

```
# myapp/admin.py
from util import admin
from myapp.models import MyModel

class MyModelAdmin(admin.ModelAdmin):
    pass
```

Low level API examples

How to get make a thumbnail in your python code:

```
from sorl.thumbnail import get_thumbnail

im = get_thumbnail(my_file, '100x100', crop='center', quality=99)
```

How to delete a file, its thumbnails as well as references in the Key Value Store:

```
from sorl.thumbnail import delete

delete(my_file)
```

CHAPTER 2

Installation & Setup

Installation

First you need to make sure to read the *Requirements*. To install sorl-thumbnail is easy:

```
pip install sorl-thumbnail
```

Or you can go to the github page

Setup

1. Add `sorl.thumbnail` to your `settings.INSTALLED_APPS`.
2. Configure your `settings`
3. If you are using the cached database key value store you need to sync the database:

   ```
   python manage.py migrate
   ```

CHAPTER 3

Requirements

Base requirements

- Python 2.7+
- Django
- *Key Value Store*
- *Image Library*

Key Value Store

sorl-thumbnail needs a Key Value Store for its operation. You can choose between a **cached database** which requires no special installation to your normal Django setup besides installing a proper cache like memcached **or** you can setup **redis** which requires a little bit more work.

Cached DB

All you need to use the cached database key value store is a database and cache setup properly using memcached. This cache needs to be really fast so **using anything else than memcached is not recomended**.

Redis

Redis is a fast key value store also suited for the job. To use the redis key value store you first need to install the redis server. After that install the redis client:

```
pip install redis
```

Image Library

You need to have an image library installed. sorl-thumbnail ships with support for Python Imaging Library, pgmagick, ImageMagick (or *GraphicsMagick*) command line tools. pgmagick are python bindings for GraphicsMagick (Magick++)‘,

The ImageMagick based engine `sorl.thumbnail.engines.convert_engine.Engine` by default calls `convert` and `identify` shell commands. You can change the paths to these tools by setting `THUMBNAIL_CONVERT` and `THUMBNAIL_IDENTIFY` respectively. Note that youneed to change these to use GraphicsMagick to `/path/to/gm convert` and `/path/to/gm identify`.

Python Imaging Library installation

Prerequisites:

- libjpeg
- zlib

Ubuntu 10.04 package installation:

```
sudo apt-get install libjpeg62 libjpeg62-dev zlib1g-dev
```

Installing Python Imaging Library using pip:

```
pip install Pillow
```

Watch the output for messages on what support got compiled in, you at least want to see the following:

```
--- JPEG support available
--- ZLIB (PNG/ZIP) support available
```

pgmagick installation

Prerequisites:

- GraphicsMagick
- Boost.Python

Ubuntu 10.04 package installation:

```
sudo apt-get install libgraphicsmagick++-dev
sudo apt-get install libboost-python1.40-dev
```

Fedora installation:

```
yum install GraphicsMagick-c++-devel
yum install boost-devel
```

Installing pgmagick using pip:

```
pip install pgmagick
```

ImageMagick installation

Ubuntu 10.04 package installation:

```
sudo apt-get install imagemagick
```

Or if you prefer GraphicsMagick:

```
sudo apt-get install graphicsmagick
```

Wand installation

Ubuntu installation:

```
apt-get install libmagickwand-dev
pip install Wand
```

CHAPTER 4

Template tags and filters

Sorl-thumbnail comes with one template tag *thumbnail* and three filters: *is_portrait*, *margin* and *resolution*. To use any of them in you templates you first need to load them:

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

thumbnail

Syntax:

```
{% thumbnail source geometry [key1=value1, key2=value2...] as var %}
{% endthumbnail %}
```

Alternative syntax using empty:

```
{% thumbnail source geometry [key1=value1, key2=value2...] as var %}
{% empty %}
{% endthumbnail %}
```

The `{% empty %}` section is rendered if the thumbnail source is resolved to an empty value or an invalid image source, you can think of it as rendering when the thumbnail becomes undefined.

Source

Source can be an ImageField, FileField, a file name (assuming default_storage), a url. What we need to know is name and storage, see how ImageFile figures these things out:

```
from django.utils.encoding import force_text

class ImageFile(BaseImageFile):
    _size = None
```

```
    def __init__(self, file_, storage=None):
        if not file_:
            raise ThumbnailError('File is empty.')
        # figure out name
        if hasattr(file_, 'name'):
            self.name = file_.name
        else:
            self.name = force_text(file_)
        # figure out storage
        if storage is not None:
            self.storage = storage
        elif hasattr(file_, 'storage'):
            self.storage = file_.storage
        elif url_pat.match(self.name):
            self.storage = UrlStorage()
        else:
            self.storage = default_storage
```

Geometry

Geometry is specified as `widthxheight`, `width` or `xheight`. Width and height are in pixels. Geometry can either be a string or resolve into a valid geometry string. Examples:

```
{% thumbnail item.image "200x100" as im %}
    <img src="{{ im.url }}">
{% endthumbnail %}

{% thumbnail item.image "200" as im %}
    <img src="{{ im.url }}">
{% endthumbnail %}

{% thumbnail item.image "x100" as im %}
    <img src="{{ im.url }}">
{% endthumbnail %}

{% thumbnail item.image geometry as im %}
    <img src="{{ im.url }}">
{% endthumbnail %}
```

If width and height are given the image is rescaled to maximum values of height and width given. Aspect ratio preserved.

Options

Options are passed on to the backend and engine, the backend generates the thumbnail filename from it and the engine can use it for processing. Option keys are not resolved in context but values are. Passing all options to the engine means that you can easily subclass an engine and create new features like rounded corners or what ever processing you like. The options described below are how they are used and interpreted in the shipped engines.

crop

This option is only used if both width and height is given. Crop behaves much like css background-position. The image is first rescaled to minimum values of height and width given, this will be equivalent to the *padding box* in the above text. After it is rescaled it will apply the cropping options. There are some differences to the css background-position:

- Only % and px are valid lengths (units)
- `noop` (No Operation) is a valid option which means there is no cropping after the initial rescaling to minimum of width and height.

There are many overlapping options here for example `center` is equivalent to `50%`. There is not a problem with that in it self but it is a bit of a problem if you will for sorl-thumbnail. Sorl-thumbnail will generate a new thumbnail for every unique source, geometry and options. This is a design choice because we want to stay flexible with the options and not interpret them anywhere else but in the engine methods. In clear words, be consistent in your cropping options if you don't want to generate unnecessary thumbnails. In case you are wondering, sorl-thumbnail sorts the options so the order does not matter, same options but in different order will generate only one thumbnail.

upscale

Upscale is a boolean and controls if the image can be upscaled or not. For example if your source is 100x100 and you request a thumbnail of size 200x200 and upscale is False this will return a thumbnail of size 100x100. If upscale was True this would result in a thumbnail size 200x200 (upscaled). The default value is `True`.

quality

Quality is a value between 0-100 and controls the thumbnail write quality. Default value is `95`.

progressive

This controls whether to save jpeg thumbnails as progressive jpegs. Default value is `True`.

orientation

This controls whether to orientate the resulting thumbnail with respect to the source EXIF tags for orientation. Default value is `True`.

format

This controls the write format and thumbnail extension. Formats supported by the shipped engines are `'JPEG'` and `'PNG'`. Default value is `'JPEG'`.

colorspace

This controls the resulting thumbnails color space, valid values are: `'RGB'` and `'GRAY'`. Default value is `'RGB'`.

padding

Padding is a boolean and controls if the image should be padded to fit the specified geometry.

If your image is `200x100`:

```
{% thumbnail image "100x100" padding=True as im %}
```

`im` will be `100x100` with white padding at the top and bottom. The color of the padding can be controlled with `padding_color` or the setting `THUMBNAIL_PADDING_COLOR` which defaults to `#ffffff`.

Images are not padded by default, but this can be changed by setting `THUMBNAIL_PADDING` to `True`.

padding_color

This is the color to use for padding the image. It defaults to `#ffffff` and can be globally set with the setting `THUMBNAIL_PADDING_COLOR`.

options

Yes this option is called `options`. This needs to be a context variable that resolves to a dictionary. This dictionary can contain multiple options, for example:

```
options = {'colorspace': 'GRAY', 'quality': 75, 'crop': 'center'}
```

You can use this option together with the other options but beware that the order will matter. As soon as the keyword `options` is encountered all the options that have a key in `options` are overwritten. Similarly, options in the `options` dict will be overwritten by options set after the options keyword argument to the thumbnail tag.

is_portrait

This filter returns True if the image height is larger than the image width. Examples:

```
{% thumbnail item.image "100x100" %}
{% if item.image|is_portrait %}
    <div class="portrait">
        <img src="{{ im.url }}">
    </div>
{% else %}
    <div class="landscape">
        <img src="{{ im.url }}">
    </div>
{% endif %}
{% endthumbnail %}

{% if item.image|is_portrait %}
    {% thumbnail item.image "100x200" crop="center" %}
        <img src="{{ im.url }}">
    {% endthumbnail %}
{% else %}
    {% thumbnail item.image "200x100" crop="center" %}
        <img src="{{ im.url }}">
    {% endthumbnail %}
{% endif %}
```

margin

Margin is a filter for calculating margins against a padding box. For example lets say you have an image `item.image` and you want to pad it vertically in a 1000x1000 box, you would simply write:

```
<div class="millxmill">
    <img src="{{ item.image.url }}" style="margin:{{ item.image|margin:"1000x1000" }}
↪">
</div>
```

The above is a rather synthetic example the more common use case is when you want boxes of images of a certain size but you do not want to crop them:

```
{% for profile in profiles %}
<div>
    {% thumbnail profile.photo "100x100" as im %}
        <img src="{{ im.url }}" style="margin:{{ im|margin:"100x100" }}">
    {% empty %}
        <img src="ghost100x100.jpg">
    {% endthumbnail %}
</div>
{% enfor %}
```

The more problematic is to get the top margin, however the margin filter outputs all values.

resolution

Resolution is a filter for obtaining alternative resolution versions of the thumbnail. Your provided resolution must be one of the `THUMBNAIL_ALTERNATIVE_RESOLUTIONS` settings values (default: no alternative resolutions)

For example, let's say you have an image `item.image` and you want to get the 2x DPI version of it. You would simply write:

```
<div class="millxmill">
    <img src="{{ item.image.url|resolution:"2x" }}">
</div>
```

CHAPTER 5

Management commands

thumbnail cleanup

```
python manage.py thumbnail cleanup
```

This cleans up the Key Value Store from stale cache. It removes references to images that do not exist and thumbnail references and their actual files for images that do not exist. It removes thumbnails for unknown images.

thumbnail clear

```
python manage.py thumbnail clear
```

This totally empties the Key Value Store of all keys that start with the `settings.THUMBNAIL_KEY_PREFIX`. It does not delete any files. The Key Value store will update when you hit the template tags, and if the thumbnails files still exist they will be used and not overwritten/regenerated. This can be useful if your Key Value Store has garbage data not dealt with by cleanup or you're switching Key Value Store backend.

thumbnail clear_delete_referenced

```
python manage.py thumbnail clear_delete_referenced
```

Equivalent to to `clear` but first it will delete all thumbnail files referenced by the Key Value Store. It is generally safe to run this if you do not reference the generated thumbnails by name somewhere else in your code. As long as all the original images still exist this will trigger a regeneration of all the thumbnails the Key Value Store knows about.

thumbnail clear_delete_all

```
python manage.py thumbnail clear_delete_all
```

Equivalent to to `clear` but afterwards it will delete all thumbnail files including any orphans not in the Key Value Store. This can be thought of as a more aggressive version of `clear_delete_referenced`. Caution should be exercised with this command if multiple Django sites (as in `SITE_ID`) or projects are using the same `MEDIA_ROOT` since this will clear out absolutely everything in the thumbnail cache directory causing thumbnail regeneration for all sites and projects. When file system storage is used, it is equivalent to `rm -rf MEDIA_ROOT + THUMBNAIL_PREFIX`

CHAPTER 6

Errors & Logging

Background

When `THUMBNAIL_DEBUG = False` errors will be suppressed if they are raised during rendering the `thumbnail` tag or raised within the included filters. This is the recommended production setting. However it can still be useful to be notified of those errors. Thus sorl-thumbnail logs errors to a logger and provides a log handler that sends emails to `settings.ADMINS`.

How to setup logging

To enable logging you need to add a handler to the 'sorl.thumbnail' logger. The following example adds the provided handler that sends emails to site admins in case an error is raised with debugging off:

```
import logging
from sorl.thumbnail.log import ThumbnailLogHandler


handler = ThumbnailLogHandler()
handler.setLevel(logging.ERROR)
logging.getLogger('sorl.thumbnail').addHandler(handler)
```

You will need to load this code somewhere in your django project, it could be in urls.py, settings.py or project/app __init__.py file for example. You could of course also provide your own logging handler.

CHAPTER 7

How sorl-thumbnail operates

When you use the `thumbnail` template tag sorl-thumbnail looks up the thumbnail in a *Key Value Store*. The key for a thumbnail is generated from its filename and storage. The thumbnail filename in turn is generated from the source and requested thumbnail size and options. If the key for the thumbnail is found in the Key Value Store, the serialized thumbnail information is fetched from it and returned. If the thumbnail key is not found there sorl-thumbnail continues to generate the thumbnail and stores necessary information in the Key Value Store. It is worth noting that sorl-thumbnail does not check if source or thumbnail exists if the thumbnail key is found in the Key Value Store.

Note: This means that if you change or delete a source file or delete the thumbnail, sorl-thumbnail will still fetch from the Key Value Store. Therefore it is important that if you delete or change a source or thumbnail file notify the Key Value Store.

If you change or delete a source or a thumbnail for some reason, you can use the `delete` method of the `ThumbnailBackend` class or subclass:

```
from sorl.thumbnail import delete

# Delete the Key Value Store reference but **not** the file.
# Use this if you have changed the source
delete(my_file, delete_file=False)

# Delete the Key Value Store reference and the file
# Use this if you want to delete the source file
delete(my_file) # delete_file=True is default
```

The `sorl.thumbnail.delete` method always deletes the input files thumbnail Key Value Store references as well as thumbnail files. You can use this method on thumbnails as well as source files. Alternatively if you have **deleted** a file you can use the management command *thumbnail cleanup*. Deleting an image using the `sorl.thumbnail.ImageField` will notify the Key Value Store to delete references to it and delete all of its thumbnail references and files, exactly like the above code example.

Why you ask? Why go through all the trouble with a Key Value Store and risk stale cache? Why not use a database to cache if you are going to do that?

The reason is speed and especially with storages other than local file storage. Checking if a file exists before serving it will cost too much. Speed is also the reason for not choosing to use a standard database for this kind of persistent caching. However sorl-thumbnail does ship with a *cached* database Key Value Store.

Note: We have to assume the thumbnail exists if the thumbnail key exists in the Key Value Store

There are bonuses. We can store meta data in the Key Value Store that would be too costly to retrieve even for local file storage. Today this meta data consists only of the image size but this could be expanded to for example EXIF data. The other bonus is that we can keep track of what thumbnails has been generated from a particular source and deleting them too when the source is deleted.

Schematic view of how things are done

CHAPTER 8

Reference

ImageFile

`ImageFile` is an image abstraction that contains useful attributes when working with images. The `thumbnail` template tag puts the generated thumbnail in context as an `ImageFile` instance. In the following example:

```
{% thumbnail item.image "100x100" as im %}
    <img src="{{ im.url }}">
{% endthumbnail %}
```

`im` will be an `ImageFile` instance.

ImageFile attributes

name

Name of the image as returned from the underlaying storage.

storage

Returns the storage instance.

width

Returns the width of the image in pixels.

x

Alias of `width`

height

Returns the height of the image in pixels.

y

Alias of `width`

ratio

Returns the image ratio (y/x) as a float

url

URL of the image url as returned by the underlaying storage.

src

Alias of `url`

size

Returns the image size in pixels as a (x, y) tuple

key

Returns a unique key based on `name` and `storage`.

ImageFile methods

exists

Returns whether the file exists as returned by the underlaying storage.

is_portrait

Returns `True` if `y > x`, else `False`

set_size

Sets the size of the image, takes an optional size tuple (x, y) as argument.

read

Reads the file as done from the underlaying storage.

write

Writes content to the file. Takes content as argument. Content is either raw data or an instance of `django.core.files.base.ContentFile`.

delete

Deletes the file from underlaying storage.

serialize

Returns a serialized version of self.

serialize_storage

Returns the `self.storage` as a serialized dot name path string.

Settings

THUMBNAIL_DEBUG

- Default: `False`

When set to `True` the `ThumbnailNode.render` method can raise errors. Django recommends that tags never raise errors in the `Node.render` method but since sorl-thumbnail is such a complex tag we will need to have more debugging available.

THUMBNAIL_BACKEND

- Default: `'sorl.thumbnail.base.ThumbnailBackend'`

This is the entry point for generating thumbnails, you probably want to keep the default one but just in case you would like to generate thumbnails filenames differently or need some special functionality you can override this and use your own implementation.

THUMBNAIL_KVSTORE

- Default: `'sorl.thumbnail.kvstores.cached_db_kvstore.KVStore'`

sorl-thumbnail needs a Key Value Store to *How sorl-thumbnail operates*. sorl-thumbnail ships with support for two Key Value Stores:

Cached DB

`sorl.thumbnail.kvstores.cached_db_kvstore.KVStore`. This is the default and preferred Key Value Store.

Features

- Fast persistent storage
- First query uses database which is slow. Successive queries are cached and if you use memcached this is very fast.
- Easy to transfer data between environments since the data is in the default database.
- If you get the database and fast cache out of sync there could be problems.

Redis

`sorl.thumbnail.kvstores.redis_kvstore.KVStore`. It requires you to install a Redis server as well as a redis python client.

Features

- Fast persistent storage
- More dependencies
- Requires a little extra work to transfer data between environments

Dbm

`sorl.thumbnail.kvstores.dbm_kvstore.KVStore`. A simple Key Value Store has no dependencies outside the standard Python library and uses the DBM modules to store the data.

Features

- No external dependencies, besides the standard library
- No extra components required, e.g., database or cache
- Specially indicated for local development environments

THUMBNAIL_KEY_DBCOLUMN

- Default `'key'`

Since MSSQL reserved the `key` name for db columns you can change this to something else using this setting.

THUMBNAIL_ENGINE

- Default: `'sorl.thumbnail.engines.pil_engine.Engine'`

This is the processing class for sorl-thumbnail. It does all the resizing, cropping or whatever processing you want to perform. sorl-thumbnail ships with three engines:

PIL

`'sorl.thumbnail.engines.pil_engine.Engine'`. This is the default engine because it is what most people have installed already. Features:

- Easy to install
- Produces good quality images but not the best
- It is fast
- Can not handle CMYK sources

Pgmagick

`'sorl.thumbnail.engines.pgmagick_engine.Engine'`. Pgmagick uses Graphics. Fatures:

- Not easy to install unless on linux, very slow to compile
- Produces high quality images
- It is a tad slow?
- Can handle CMYK sources

ImageMagick / GraphicsMagick

`'sorl.thumbnail.engines.convert_engine.Engine'`. This engine uses the ImageMagick `convert` or GraphicsMagic `gm convert` command. Features:

- Easy to install
- Produces high quality images
- It is pretty fast
- Can handle CMYK sources
- It is a command line command, that is less than ideal,

Wand

`'sorl.thumbnail.engines.wand_engine.Engine'`. This engine uses Wand, a ctypes-based simple ImageMagick binding for Python. Features:

- Easy to install
- Produces high quality images
- Can handle CMYK sources
- Works on Python 2.6, 2.7, 3.2, 3.3, and PyPy

THUMBNAIL_CONVERT

- Default `'convert'`

Path to convert command, use `'gm convert'` for GraphicsMagick. Only applicable for the convert Engine.

THUMBNAIL_IDENTIFY

- Default `'identify'`

Path to identify command, use `'gm identify'` for GraphicsMagick. Only applicable for the convert Engine.

THUMBNAIL_STORAGE

- Default: `settings.DEFAULT_FILE_STORAGE`

The storage class to use for the generated thumbnails.

THUMBNAIL_REDIS_DB

- Default: `0`

The Redis database. Only applicable for the Redis Key Value Store

THUMBNAIL_REDIS_PASSWORD

- Default: `''`

The password for Redis server. Only applicable for the Redis Key Value Store

THUMBNAIL_REDIS_HOST

- Default: `'localhost'`

The host for Redis server. Only applicable for the Redis Key Value Store

THUMBNAIL_REDIS_PORT

- Default: `6379`

The port for Redis server. Only applicable for the Redis Key Value Store

THUMBNAIL_DBM_FILE

- Default: `thumbnail_kvstore`

Filename of the DBM database. Depending on the DBM engine selected by your Python installation, this will be used as a prefix because multiple files may be created. This can be an absolute path.

THUMBNAIL_DBM_MODE

- Default: `0x644`

Permission bits to use when creating new DBM files

THUMBNAIL_CACHE_TIMEOUT

- Default: `3600 * 24 * 365 * 10`

Cache timeout for Cached DB Key Value Store in seconds. You should probably keep this at maximum or `None` if your caching backend can handle that as infinite. Only applicable for the Cached DB Key Value Store.

THUMBNAIL_CACHE

- Default: `'default'`

Cache configuration for Cached DB Key Value Store. Defaults to the `'default'` cache but some applications might have multiple cache clusters.

THUMBNAIL_KEY_PREFIX

- Default: `'sorl-thumbnail'`

Key prefix used by the key value store.

THUMBNAIL_PREFIX

- Default: `'cache/'`

The generated thumbnails filename prefix.

THUMBNAIL_FORMAT

- Default: `'JPEG'`

Default image format, supported formats are: `'JPEG'`, `'PNG'`. This also implicitly sets the filename extension. This can be overridden by individual options.

THUMBNAIL_PRESERVE_FORMAT

- Default: `False`

If `True`, the format of the input file will be preserved. If `False`, `THUMBNAIL_FORMAT` will be used.

THUMBNAIL_COLORSPACE

- Default: `'RGB'`

Default thumbnail color space, engines are required to implement: `'RGB'`, `'GRAY'` Setting this to None will keep the original color space. This can be overridden by individual options.

THUMBNAIL_UPSCALE

- Default: `True`

Should we upscale by default? `True` means we upscale images by default. `False` means we don't. This can be overridden by individual options.

THUMBNAIL_QUALITY

- Default: `95`

Default thumbnail quality. A value between 0 and 100 is allowed. This can be overridden by individual options.

THUMBNAIL_PROGRESSIVE

- Default: `True`

Saves jpeg thumbnails as progressive jpegs. This can be overridden by individual options.

THUMBNAIL_ORIENTATION

- Default: `True`

Orientate the thumbnail with respect to source EXIF orientation tag

THUMBNAIL_DUMMY

- Default: `False`

This is a very powerful option which came from real world frustration. The use case is when you want to do development on a deployed project that has image references in its database. Instead of downloading all the image files from the server hosting the deployed project and all its thumbnails we just set this option to `True`. This will generate placeholder images for all thumbnails missing input source.

THUMBNAIL_DUMMY_SOURCE

- Default `http://dummyimage.com/%(width)sx%(height)s`

This is the generated thumbnail whensource of the presented thumbnail. Width and Height is passed to the string for formatting. Other options are for example:

- `http://placehold.it/%(width)sx%(height)s`
- `http://placekitten.com/%(width)s/%(height)s`

THUMBNAIL_DUMMY_RATIO

- Default: `1.5`

This value sets an image ratio to all thumbnails that are not defined by width **and** height since we cannot determine from the file input (since we don't have that).

THUMBNAIL_ALTERNATIVE_RESOLUTIONS

- Default: `[]`
- Example: `[1.5, 2]`

This value enables creation of additional high-resolution ("Retina") thumbnails for every thumbnail. Resolution multiplicators, e.g. value 2 means for every thumbnail of regular size x*y, additional thumbnail of 2x*2y size is created.

THUMBNAIL_FILTER_WIDTH

- Default: `500`

This value sets the width of thumbnails inserted when running filters one texts that regex replaces references to images with thumbnails.

THUMBNAIL_URL_TIMEOUT

- Default: `None`

This value sets the timeout value in seconds when retrieving a source image from a URL. If no timeout value is specified, it will wait indefinitely for a response.

CHAPTER 9

Contributing

Feel free to create a new Pull request if you want to propose a new feature or fix a bug. If you need development support or want to discuss with other developers, join us in the channel #sorl-thumnbnail at freenode.net

irc://irc.freenode.net/#sorl-thumbnail

Running testsuit

For occasional developers we recommend using Travis CI to run testsuit, for those who want to run tests locally, read on.

Since sorl-thumbnail supports a variety of image backends, python and Django versions, we provide an easy way to test locally across all of them. We use Vagrant for simple interaction with virtual machines and tox for managing python virtual environments.

Some dependencies like pgmagick takes a lot of time to compiling. To speed up your vagrant box you can edit Vagrant file with mem and cpu or simply install vagrant-faster. The resulting .tox folder containing all virtualenvs requires ~

- Install Vagrant
- `cd` in your source directory
- Run `vagrant up` to prepare VM. It will download Ubuntu image and install all necessary dependencies.
- Run `vagrant ssh` to log in the VM
- Launch all tests via `tox` (will take some time to build envs first time)

To run only tests against only one configuration use `-e` option:

```
tox -e py34-django16-pil
```

Py34 stands for python version, 1.6 is Django version and the latter is image library. For full list of tox environments, see `tox.ini`

You can get away without using Vagrant if you install all packages locally yourself, however, this is not recommended.

Sending pull requests

1. Fork the repo:

```
git@github.com:mariocesar/sorl-thumbnail.git
```

2. Create a branch for your specific changes:

```
$ git checkout master
$ git pull
$ git checkout -b feature/foobar
```

   To simplify things, please, make one branch per issue (pull request). It's also important to make sure your branch is up-to-date with upstream master, so that maintainers can merge changes easily.

3. Commit changes. Please update docs, if relevant.
4. Don't forget to run tests to check than nothing breaks.
5. Ideally, write your own tests for new feature/bug fix.
6. Submit a pull request.

CHAPTER 10

News

Next Release

A final 12.0 version is planned for end of 2013.

Announces

2013-11-12

@mariocesar: We have a new team of mantainers for sorl-thumbnail. I wan't to encourage all developers that have fixes, forks and new features to talk in the irc channel of the project. irc://freenode.net/#sorl-thumbnail

2013-11-09

@nikko: Unfortunately I no longer work with Django so my motivation to keep developing for sorl-thumbnail is very low. If you are interested in taking over this project please send an email to mikko@aino.se explaing why you are the perfect fit for owning this project.

CHAPTER 11

Changelog

12.0 (TBA)

- [Feature] #145 Python 3 support
- [Feature] #165 Django 1.5-1.6 support
- [Feature] #308 Django 1.7
- [Feature] #290 New DBM based Key-Value Store
- [Feature] Cropbox option
- [Feature] Rounded corners
- [Feature] Vagrant testing
- [Feature] #89 `THUMBNAIL_URL_TIMEOUT` setting for retrieving an image with a URL
- [Feature] #97 New style tag `<img src="{% thumbnail obj.image "200x150" crop="center" %}"/>`
- [Feature] Blur support for (PIL) engine
- [Feature] `background_margin` filter
- [Feature] #135 Ability to preserve file format
- [Feature] #159 Wand engine
- [Feature] #178 Improved error logging in templates
- [Feature] #176 Custom CACHE storage
- [Feature] #191 Added text filters `markdown_thumbnails` and `html_thumbnails`
- [Feature] #187 Padding around the thumbnail
- [Feature] #201 Flatten images (imagemagick)
- [Fix] #73 Multipage PDFs and animated GIF support (imagemagick)

- [Fix] #78 Bad resize when upscale is off and image is small
- [Fix] Multiple docs fixes
- [Fix] #82 `ValueError` errors in orientation (imagemagick)
- [Fix] #83 `open()` calls on windows (imagemagick)
- [Fix] #92 Improved interaction with S3 storage engine
- [Fix] #94 Display thumbnail if it exists, even if `THUMBNAIL_DUMMY` is `True`
- [Fix] #98 #137 #113 Exif errors (PIL)
- [Fix] #129 #214 Support for very large images (PIL)
- [Fix] #39 `get_thumbnail` doesn't respect `THUMBNAIL_DUMMY` setting
- [Fix] Thumbnail error occurring when file is blank (40fe1b0)
- [Fix] #148 Error in thumbnail clear command, when storage is empty
- [Fix] #139 Proper UrlStorage url to prevent HTTP Error 505
- [Fix] #162 Sporadic IntegrityError when calling get_thumbnail
- [Fix] #116 KeyError when Image file raw data is not a valid image
- [Fix] #186 Better expetion handling for `AdminImageWidget`
- [Fix] #192 Fixes photo desaturation issue (PIL)
- [Fix] #203 Remove check if file exists from the templatetag code
- [Fix] #213 Fixed descriptor leak (imagemagick)
- [Fix] #216 #217 Fixed OSError handling (PIL)
- [Fix] #304 #14 Fix AdminWidget
- [Fix] #274 Fix issue with transparent PNGs: IOError("cannot use transparency for this mode")
- [Fix] #265 Better support for LA mode images in PIL engine
- [Fix] #261 THUMBNAIL_DEBUG has False as default

CHAPTER 12

Indices and tables

- genindex
- modindex
- search