src Documentation

Release 0.2.5

Alisue

September 29, 2015

Contents

1 **django-thumbnailfield** 1

2 **Install** 3

3 **Prepare to use** 5

4 **Example mini blog app** 7

5 **How to use custom process method** 9

6 **Settings** 11

7 **API documentation** 13

7.1 thumbnailfield Package 13

8 **Indices and tables** 21

Python Module Index 23

CHAPTER 1

django-thumbnailfield

Author Alisue <lambdalisue@hashnote.net>

Supported python versions 2.6, 2.7, 3.2, 3.3, 3.4

Supported django versions 1.3 - 1.6 and 1.7c1

django-thumbnailfield is a enhanced ImageField of Django. It has the follwing features

- Using Django storage system to store the image (Not like other Thumbnail library)
- Automatically remove previous file from storage
- Automatically generate thumbnails
- Automatically remove generated previous thumbnail files from storage
- Easy to use custom method to generate thumbnails

CHAPTER 2

Install

```
sudo pip install django-thumbnailfield
```

CHAPTER 3

Prepare to use

1. Append 'thumbnailfield' to `INSTALLED_APPS`
2. Set `MEDIA_ROOT` correctly. For example:

```
MEDIA_ROOT = os.path.join(os.path.dirname(__file__), '../static')
```

CHAPTER 4

Example mini blog app

models.py:

```
from django.db import models
from django.contrib.auth.models import User

from thumbnailfield.fields import ThumbnailField

class Entry(models.Model):
    PUB_STATES = (
        ('public', 'public entry'),
        ('protected', 'login required'),
        ('private', 'secret entry'),
    )
    pub_state = models.CharField('publish status', choices=PUB_STATES)
    title = models.CharField('title', max_length=140)
    body = models.TextField('body')

    #
    # This is a usage of ThumbnailField.
    # You have to set ``patterns`` to generate thumbnails
    #
    thumbnail = ThumbnailField(_('thumbnail'), upload_to='img/thumbnails', null=True, blank=True,
        pil_save_options={
            # Options of PIL Image.save() method.
            # e.g. quality control
            'quality': 100,
        },
        patterns={
            # Pattern Format:
            #   <Name>: (
            #   (<square_size>,),        # with defautl process_method
            #   (<width>, <height>,),    # with default process_method
            #   (<width>, <height>, <method or method_name>),
            #   (<width>, <height>, <method or method_name>, <method options>),
            #   )
            #
            # If Name is ``None`` that mean original image will be processed
            # with the pattern
            #
            # Convert original image to sepia and resize it to 800x400 (original
            # size is 804x762)
            None: ((None, None, 'sepia'), (800, 400, 'resize')),
            # Create 640x480 resized thumbnail as large.
```

```
            'large': ((640, 480, 'resize'),),
            # Create 320x240 cropped thumbnail as small. You can write short
            # pattern if the number of appling pattern is 1
            'small': (320, 240, 'crop', {'left': 0, 'upper': 0}),
            # Create 160x120 thumbnail as tiny (use default process_method to
            # generate)
            'tiny': (160, 120),
            #
            # These thumbnails are not generated while accessed. These can be
            # accessed with the follwoing code::
            #
            #   entry.thumbnail.large
            #   entry.thumbnail.small
            #   entry.thumbnail.tiny
            #
            #   # shortcut properties
            #   entry.thumbnail.large_file  # as entry.thumbnail.large.file
            #   entry.thumbnail.large_path  # as entry.thumbnail.large.path
            #   entry.thumbnail.large_url   # as entry.thumbnail.large.url
            #   entry.thumbnail.large.size  # as entry.thumbnail.large.size
            #
        })
    # ...
```

entry_detail.html:

```
<html>
<head>
    <title>django-thumbnailfield example</title>
</head>
<body>
    <dl>
        <dt>Original</dt>
        <dd><img src="{{ MEDIA_URL }}{{ object.thumbnail }}"></dd>
        <dt>Thumbnail "large"</dt>
        <dd><img src="{{ MEDIA_URL }}{{ object.thumbnail.large }}"></dd>
        <dt>Thumbnail "small"</dt>
        <dd><img src="{{ MEDIA_URL }}{{ object.thumbnail.small }}"></dd>
        <dt>Thumbnail "tiny"</dt>
        <dd><img src="{{ MEDIA_URL }}{{ object.thumbnail.tiny }}"></dd>
    </dl>
</body>
</html>
```

CHAPTER 5

How to use custom process method

Create your own custom process method like below:

```
from django.core.exceptions import ImproperlyConfigured
from thumbnailfield.process_methods import get_sepia_image
from thumbnailfield.process_methods import get_cropped_image

def get_sepia_and_cropped_image(img, width, height, **options):
    # do something with img
    img = get_sepia_image(img, None, None, **options)
    img = get_cropped_image(img, width, height, **options)
    return img
def _sepia_and_cropped_error_check(f, img, width, height, **options):
    # do some error check
    if 'left' not in options:
        raise ImproperlyConfigured(f, "'left' is required")
    if 'upper' not in options:
        raise ImproperlyConfigured(f, "'upper' is required")
# Apply error check function
# Error check is recommended if your process method required any options
# otherwise just forget about this.
get_sepia_and_cropped_image.error_check = _sepia_and_cropped_error_check
```

Use defined method in pattern like below:

```
# models.py
# ...
thumbnail = ThumbnailField('thumbnail', upload_to='thumbnails', patterns = {
        'large': (400, 500, get_sepia_and_cropped_image, {'left': 0, 'upper': 0})
    }
# ...
```

Or define the method in THUMBNAILFIELD_PROCESS_METHOD_TABLE and use as a string anme:

```
# settings.py
from thumbnailfield import DEFAULT_PROCESS_METHOD_TABLE
THUMBNAILFIELD_PROCESS_METHOD_TABLE = DEFAULT_PROCESS_METHOD_TABLE
THUMBNAILFIELD_PROCESS_METHOD_TABLE['sepia_and_crop'] = get_sepia_and_cropped_image

# models.py
# ...
thumbnail = ThumbnailField('thumbnail', upload_to='thumbnails', patterns = {
        'large': (400, 500, 'sepia_and_crop', {'left': 0, 'upper': 0})
    }
# ...
```

If `None` is specified, that mean do nothing.

```
# models.py # ... thumbnail = ThumbnailField('thumbnail', upload_to='thumbnails', patterns = {
        'original': None,
    }
# ...
```

CHAPTER 6

Settings

THUMBNAILFIELD_REMOVE_PREVIOUS Remove previous files (include original file) when new file is applied to the ThumbnailField.

Default: `False`

THUMBNAILFIELD_DEFAULT_PROCESS_METHOD Used when no process_method is applied in process pattern.

Default: `thumbnail`

THUMBNAILFIELD_DEFAULT_PROCESS_OPTIONS Used when no process_options is applied in process pattern.

Default: `{'resample': Image.ANTIALIAS}`

THUMBNAILFIELD_FILENAME_PATTERN Used to determine thumbnail filename. `root`, `filename`, `name` and `ext` is passed to the string. The generated filename of the thumbnail named 'large' of '/some/where/test.png' will be `/some/where/test.large.png` in default.

Default: `r"%(root)s/%(filename)s.%(name)s.%(ext)s"`

THUMBNAILFIELD_PROCESS_METHOD_TABLE Used to determine process method from string name. The key of this dictionary is a name of the method and value is a method.

`thumbnail`, `resize`, `crop`, `grayscale` and `sepia` are defined as default.

Default: See `thumbnailfield.__init__.DEFAULT_PROCESS_METHOD_TABLE`

THUMBNAILFIELD_DEFAULT_PIL_SAVE_OPTIONS Options used in PIL image save method.

Default: `{}`

CHAPTER 7

API documentation

7.1 thumbnailfield Package

7.1.1 compatibility Module

Compatibility module

7.1.2 conf Module

class thumbnailfield.conf.**ThumbnailFieldConf**(***kwargs*)

Bases: appconf.base.AppConf

DEFAULT_PIL_SAVE_OPTIONS = {}

DEFAULT_PROCESS_METHOD = 'thumbnail'

DEFAULT_PROCESS_OPTIONS = {'resample': 1}

FILENAME_PATTERN = '%(root)s/%(filename)s.%(name)s.%(ext)s'

class Meta

Bases: object

ThumbnailFieldConf.**PROCESS_METHOD_TABLE = {'sepia': <function get_sepia_image at 0x7f7d7092c668>, 'crop':**

ThumbnailFieldConf.**REMOVE_PREVIOUS = False**

7.1.3 fields Module

Model fields of ThumbnailField

class thumbnailfield.fields.**ThumbnailField**(*verbose_name=None*, *name=None*, *width_field=None*, *height_field=None*, *patterns=None*, *pil_save_options=None*, ***kwargs*)

Bases: django.db.models.fields.files.ImageField

Enhanced ImageField

ThumbnailField has the follwoing features

- Automatically remove previous file
- Automatically generate thumbnail files

•Automatically remove generated previous thumbnail files

attr_class
alias of *ThumbnailFieldFile*

description = <django.utils.functional.__proxy__ object>

descriptor_class
alias of *ThumbnailFileDescriptor*

class thumbnailfield.fields.**ThumbnailFieldFile**(**args*, ***kwargs*)
Bases: `django.db.models.fields.files.ImageFieldFile`

Enhanced ImageFieldFile

This FieldFile contains thumbnail ImageFieldFile instances and these thumbnails are automatically generate when accessed

_get_thumbnail_filename -- get thumbnail filename

_get_image -- get PIL image instance

_get_thumbnail -- get PIL image instance of thumbnail

_get_thumbnail_file -- get ImageFieldFile instance of thumbnail

_create_thumbnail -- create PIL image instance of thumbnail

_create_thumbnail_file -- create ImageFieldFile instance of thumbnail

_update_thumbnail_file -- update thumbnail file and return
ImageFieldFile instance

_remove_thumbnail_file -- remove thumbanil file from storage

iter_pattern_name -- return iterator of pattern name

get_pattern_name -- return list of pattern name

iter_thumbnail_filenames -- return iterator of thumbnail filename

get_thumbnail_filenames -- return list of thumbnail filename

iter_thumbnail_files -- return iterator of thumbnail file

get_thumbnail_files -- return list of thumbnail file

update_thumbnail_files -- update thumbnail files in storage

remove_thumbnail_files -- remove thumbnail files from storage

delete(*save=True*)

get_pattern_names()
return list of thumbnail pattern names

get_thumbnail_filenames()
return list of thumbnail filenames

get_thumbnail_files()
return list of thumbnail files

iter_pattern_names()
return iterator of thumbnail pattern names

iter_thumbnail_filenames()
return iterator of thumbnail filenames

iter_thumbnail_files()
return iterator of thumbnail files

remove_thumbnail_files(*save=True*)
remove thumbnail files from storage

Attribute: save – If true, the model instance of this field will be saved.

save(*name*, *content*, *save=True*)

update_thumbnail_files()
update thumbanil files of storage

class thumbnailfield.fields.**ThumbnailFileDescriptor**(*field*)
Bases: `django.db.models.fields.files.ImageFileDescriptor`

Enhanced ImageFileDescriptor

Just like the ImageFileDescriptor, but for ThumbnailField. The only difference is removing previous Image and Thumbnails from storage when the value has changed.

7.1.4 models Module

7.1.5 process_methods Module

Builtin ThumbnailField process methods

thumbnailfield.process_methods.**get_cropped_image**(*img*, *width*, *height*, *left*, *upper*, ***options*)
get cropped image

Attribute: img – PIL image instance width – width of thumbnail height – height of thumbnail left – left point of thumbnail upper – upper point of thumbnail kwargs – Options used in PIL thumbnail method

Usage::

```
>>> from thumbnailfield.compatibility import Image
>>> img = Image.new('RGBA', (1000, 800))
>>> thumb = get_cropped_image(img, 100, 100, 0, 0)
>>> assert thumb.size[0] == 100
>>> assert thumb.size[1] == 100
```

thumbnailfield.process_methods.**get_grayscale_image**(*img*, *width*, *height*, ***options*)
get grayscale image

Attribute: img – PIL image instance width – width of thumbnail height – height of thumbnail kwargs – Options used in PIL thumbnail method

Usage::

```
>>> from thumbnailfield.compatibility import Image
>>> img = Image.new('RGBA', (1000, 800))
>>> thumb = get_grayscale_image(img, 100, 100)
```

thumbnailfield.process_methods.**get_resized_image**(*img*, *width*, *height*, *force=False*, ***options*)
get resized image

Attribute: img – PIL image instance width – width of thumbnail height – height of thumbnail kwargs – Options used in PIL thumbnail method

Usage::

```
>>> from thumbnailfield.compatibility import Image
>>> img = Image.new('RGBA', (1000, 800))
>>> thumb = get_resized_image(img, 100, 100)
>>> assert thumb.size[0] == 100
>>> assert thumb.size[1] == 100
```

`thumbnailfield.process_methods.`**`get_sepia_image`**(*img*, *width*, *height*, ***options*)

get sepia image

Attribute: img – PIL image instance width – width of thumbnail height – height of thumbnail kwargs – Options used in PIL thumbnail method

Usage::

```
>>> from thumbnailfield.compatibility import Image
>>> img = Image.new('RGBA', (1000, 800))
>>> thumb = get_sepia_image(img, 100, 100)
```

`thumbnailfield.process_methods.`**`get_thumbnail_image`**(*img*, *width*, *height*, ***options*)

get thumbnail image

Attribute: img – PIL image instance width – width of thumbnail height – height of thumbnail kwargs – Options used in PIL thumbnail method

Usage::

```
>>> from thumbnailfield.compatibility import Image
>>> img = Image.new('RGBA', (1000, 800))
>>> thumb = get_thumbnail_image(img, 100, 100)
>>> assert thumb.size[0] == 100
>>> assert thumb.size[1] == 80
```

7.1.6 `utils` Module

Utilities of ThumbnailField

`thumbnailfield.utils.`**`get_content_file`**(*img*, *file_fmt*, ***kwargs*)

get ContentFile from PIL image instance with file_fmt

`img -- PIL Image instance`

`file_fmt -- Saved image format`

PNG, JPEG, ...

`kwargs -- Options used in PIL image save method`

Usage::

```
>>> from thumbnailfield.compatibility import Image
>>> from django.core.files.base import ContentFile
>>> img = Image.new('RGBA', (100, 100))
>>> cf = get_content_file(img, 'PNG')
>>> assert isinstance(cf, ContentFile)
```

`thumbnailfield.utils.`**`get_fileformat_from_filename`**(*filename*)

get fileformat from filename

`filename -- filename used to guess fileformat`

Usage::

```
>>> assert get_fileformat_from_filename("test.png") == "PNG"
>>> assert get_fileformat_from_filename("test.jpg") == "JPEG"
>>> assert get_fileformat_from_filename("test.jpe") == "JPEG"
>>> assert get_fileformat_from_filename("test.jpeg") == "JPEG"
>>> assert get_fileformat_from_filename("test.gif") == "GIF"
>>> assert get_fileformat_from_filename("test.tif") == "TIFF"
>>> assert get_fileformat_from_filename("test.tiff") == "TIFF"
>>> assert get_fileformat_from_filename("test.bmp") == "BMP"
>>> assert get_fileformat_from_filename("test.dib") == "BMP"
>>> assert get_fileformat_from_filename(
...     "/some/where/test.png") == "PNG"
```

`thumbnailfield.utils.`**`get_processed_image`**(*f*, *img*, *patterns*)

process PIL image with pattern attribute

f -- ThumbnailFieldFile instance

img -- PIL Image instance

patterns -- Process patterns

pattern format is shown below:

```
# Use default process_method with default process options and same
# width, height
pattern = [square_size]
# Use default process_method with default process options and width,
# height
pattern = [width, height]
# Use process_method with default process options and width, height
pattern = [width, height, process_method]
# Use process_method with process_options with widht, height
pattern = [width, height, process_method, process_options]
```

default process_method and process_options are configured in settings.py as:

```
THUMBNAILFIELD_DEFAULT_PROCESS_METHOD = 'thumbnail'
THUMBNAILFIELD_DEFAULT_PROCESS_OPTIONS = {'filter': Image.ANTIALIAS}
```

`thumbnailfield.utils.`**`get_thumbnail_filename`**(*path*, *name*, *pattern=None*)

get thumbnail filename with name and pattern

path -- original path

name -- thumbnail name

pattern -- file name generation pattern (default = settings.THUMBNAILFIELD_FILENAME_PATTERN)

Usage::

```
>>> path = "/some/where/test.png"
>>> name = "small"
>>> pattern = r"%(root)s/%(filename)s.%(name)s.%(ext)s"
>>> thumb_filename = get_thumbnail_filename(path, name, pattern)
>>> assert thumb_filename == "/some/where/test.small.png"
```

`thumbnailfield.utils.`**`save_to_storage`**(*img*, *storage*, *filename*, *overwrite=False*, ***kwargs*)

save PIL image instance to Django storage with filename

img -- PIL Image instance

storage -- Django storage instance

filename -- filename

overwrite -- If true, delete existing file first

kwargs -- Options used in PIL image save method

Usage::

```
>>> from thumbnailfield.compatibility import Image
>>> from django.core.files.storage import FileSystemStorage
>>> img = Image.new('RGBA', (100, 100))
>>> storage = FileSystemStorage()
>>> filename = 'test.png'
>>> # save image to storage
>>> filename_ = save_to_storage(img, storage, filename)
>>> # file exists
>>> assert storage.exists(filename_)
>>> # resave
>>> filename_ = save_to_storage(img, storage, filename)
>>> # used different filename
>>> assert filename != filename_
>>> storage.delete(filename_)
>>> # overwrite the file
>>> filename_ = save_to_storage(img, storage, filename, overwrite=True)
>>> assert filename == filename_
>>> storage.delete(filename)
```

7.1.7 Subpackages

tests Package

tests Package

thumbnailfield.tests.**suite**()

models Module

class thumbnailfield.tests.models.**Entry**(*id*, *title*, *body*, *thumbnail*)

Bases: `django.db.models.base.Model`

exception DoesNotExist

Bases: `django.core.exceptions.ObjectDoesNotExist`

exception Entry.**MultipleObjectsReturned**

Bases: `django.core.exceptions.MultipleObjectsReturned`

Entry.**objects = <django.db.models.manager.Manager object>**

test_doctests Module

thumbnailfield.tests.test_doctests.**load_tests**(*loader*, *tests*, *ignore*)

`test_thubmanilfield` Module

class `thumbnailfield.tests.test_thubmanilfield.`**`ThumbnailFieldTestCase`**(*methodName='runTest'*)

Bases: `django.test.testcases.TestCase`

`test_thumbnailfield_creation`()

`test_thumbnailfield_modification`()

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

t

thumbnailfield.compatibility, 13
thumbnailfield.conf, 13
thumbnailfield.fields, 13
thumbnailfield.models, 15
thumbnailfield.process_methods, 15
thumbnailfield.tests, 18
thumbnailfield.tests.models, 18
thumbnailfield.tests.test_doctests, 18
thumbnailfield.tests.test_thubmanilfield, 19
thumbnailfield.utils, 16

A

attr_class (thumbnailfield.fields.ThumbnailField attribute), 14

D

DEFAULT_PIL_SAVE_OPTIONS (thumbnailfield.conf.ThumbnailFieldConf attribute), 13
DEFAULT_PROCESS_METHOD (thumbnailfield.conf.ThumbnailFieldConf attribute), 13
DEFAULT_PROCESS_OPTIONS (thumbnailfield.conf.ThumbnailFieldConf attribute), 13
delete() (thumbnailfield.fields.ThumbnailFieldFile method), 14
description (thumbnailfield.fields.ThumbnailField attribute), 14
descriptor_class (thumbnailfield.fields.ThumbnailField attribute), 14

E

Entry (class in thumbnailfield.tests.models), 18
Entry.DoesNotExist, 18
Entry.MultipleObjectsReturned, 18

F

FILENAME_PATTERN (thumbnailfield.conf.ThumbnailFieldConf attribute), 13

G

get_content_file() (in module thumbnailfield.utils), 16
get_cropped_image() (in module thumbnailfield.process_methods), 15
get_fileformat_from_filename() (in module thumbnailfield.utils), 16
get_grayscale_image() (in module thumbnailfield.process_methods), 15
get_pattern_names() (thumbnailfield.fields.ThumbnailFieldFile method), 14
get_processed_image() (in module thumbnailfield.utils), 17
get_resized_image() (in module thumbnailfield.process_methods), 15
get_sepia_image() (in module thumbnailfield.process_methods), 16
get_thumbnail_filename() (in module thumbnailfield.utils), 17
get_thumbnail_filenames() (thumbnailfield.fields.ThumbnailFieldFile method), 14
get_thumbnail_files() (thumbnailfield.fields.ThumbnailFieldFile method), 14
get_thumbnail_image() (in module thumbnailfield.process_methods), 16

I

iter_pattern_names() (thumbnailfield.fields.ThumbnailFieldFile method), 14
iter_thumbnail_filenames() (thumbnailfield.fields.ThumbnailFieldFile method), 14
iter_thumbnail_files() (thumbnailfield.fields.ThumbnailFieldFile method), 14

L

load_tests() (in module thumbnailfield.tests.test_doctests), 18

O

objects (thumbnailfield.tests.models.Entry attribute), 18

P

PROCESS_METHOD_TABLE (thumbnailfield.conf.ThumbnailFieldConf attribute),

13

R

REMOVE_PREVIOUS (thumbnailfield.conf.ThumbnailFieldConf attribute), 13
remove_thumbnail_files() (thumbnailfield.fields.ThumbnailFieldFile method), 15

S

save() (thumbnailfield.fields.ThumbnailFieldFile method), 15
save_to_storage() (in module thumbnailfield.utils), 17
suite() (in module thumbnailfield.tests), 18

T

test_thumbnailfield_creation() (thumbnailfield.tests.test_thubmanilfield.ThumbnailFieldTestCase method), 19
test_thumbnailfield_modification() (thumbnailfield.tests.test_thubmanilfield.ThumbnailFieldTestCase method), 19
ThumbnailField (class in thumbnailfield.fields), 13
thumbnailfield.compatibility (module), 13
thumbnailfield.conf (module), 13
thumbnailfield.fields (module), 13
thumbnailfield.models (module), 15
thumbnailfield.process_methods (module), 15
thumbnailfield.tests (module), 18
thumbnailfield.tests.models (module), 18
thumbnailfield.tests.test_doctests (module), 18
thumbnailfield.tests.test_thubmanilfield (module), 19
thumbnailfield.utils (module), 16
ThumbnailFieldConf (class in thumbnailfield.conf), 13
ThumbnailFieldConf.Meta (class in thumbnailfield.conf), 13
ThumbnailFieldFile (class in thumbnailfield.fields), 14
ThumbnailFieldTestCase (class in thumbnailfield.tests.test_thubmanilfield), 19
ThumbnailFileDescriptor (class in thumbnailfield.fields), 15

U

update_thumbnail_files() (thumbnailfield.fields.ThumbnailFieldFile method), 15