# django-undermythumb Documentation

***Release***



September 10, 2011

# CONTENTS

`undermythumb` is a simple thumbnailing library for Django with a twist: its fields can optionally fall back to thumbnails from another field.

This app is currently used around Pitchfork, and is under active development.

Issues can be reported here.

CHAPTER

ONE

# WHY ANOTHER THUMBNAILER?

At Pitchfork, we needed a simple way to cut thumbnails, place them on the field's storage, do without fancy renderers we'd never use, and give editors a *simple* way to override auto-generated thumbnails without littering our templates with unnecessary logic.

CHAPTER

TWO

# BASIC EXAMPLE

In `models.py`:

```
from django.db import models

from undermythumb.fields import (ImageWithThumbnailsField,
                                 ImageFallbackField)
from undermythumb.renderers import CropRenderer


class BlogPost(models.Model):
    title = models.CharField(max_length=100)

    # an image with thumbnails
    artwork = ImageWithThumbnailsField(
         max_length=255,
         upload_to='artwork/',
         thumbnails=(('homepage_image', CropRenderer(300, 150)),
                     ('pagination_image', CropRenderer(150, 75))),
         help_text='Blog post header artwork.')

    # an override field, capable of rolling up a dotted path
    # if the field is empty.
    # ``fallback_path`` should point to something that returns an ``ImageFieldFile``,
    # or anything that has an ``url`` attribute, and displays an image.
    #
    # under the hood, this is just an ImageField.
    homepage_image = ImageFallbackField(
        fallback_path='artwork.thumbnails.homepage_image',
        upload_to='artwork/',
        help_text='Optional override for "homepage_image" thumbnail.')

    def __unicode__(self):
        return self.title
```

In a template, where `object` is an instance of `BlogPost`:

```
<!-- does the job: auto-generated thumbnail from "artwork" -->
<img src="{{ object.artwork.thumbnails.homepage_image.url }}" />

<!-- smarter: value of "homepage_image", or, if empty, value of "artwork.thumbnails.homepage_image" -
<img src="{{ object.homepage_image.url }}" />
```

CHAPTER

THREE

# FALLING BACK? WHAT?

Sometimes system generated thumbnails are ugly, or need to be overridden. In this case, offering an override field allows designers and editors a way to control the thumbnail used across your site.

Sometimes, default thumbnails are perfect, and no additional work is required.

The easiest way to account for both scenarios is to have a single field that allows an optional override, and can fall back to another image if no override is needed.

This way,

- Content editors need only cut thumbnails when what the system's generated
- Developers need only reference one thumbnail field in templates and code

CHAPTER

FOUR

# DOCUMENTATION

## 4.1 Installation

Installing via PyPI is recommended, unless you're planning on some hacking.

1. Install via PyPI:

   ```
   pip install django-undermythumb
   ```

2. Add `'undermythumb'` to your `INSTALLED_APPS` setting:

   ```
   INSTALLED_APPS = (
       # ...
       'undermythumb',
   )
   ```

## 4.2 Fields

The following fields (and concepts) are offered by `undermythumb`.

### 4.2.1 Available fields

`undermythumb` comes with two fields, each with specific use cases.

#### `ImageWithThumbnailsField`

The name says it all – this field is used to accept an image and cut thumbnails. Thumbnails are stored using whatever storage is available to the field.

##### Defining thumbnails

Thumbnails are defined as a tuple of tuples passed as `thumbnails` to the field.

Each tuple is defined as `(thumbnail key, renderer instance)`. The thumbnail key becomes the thumbnail's key when using the field's `thumbnails` attribute to display a thumbnail, and part of the filename generated on creation.

Example:

```
artwork = ImageWithThumbnailsField(
    thumbnails = (('related_content', CropRenderer(150, 150)), ),
    upload_to='artwork/',
)
```

In the example above, `artwork` has one thumbnail: `related_content`.

In a template (or Python code), you can access `related_content` via `thumbnails`:

```
{# assuming 'object' is an instance of your model #}
{{ object.artwork.thumbnails.related_content.url }}
```

---

**Note:** For a complete list of renderers, see: *List of available renderers*.

---

#### ImageFallbackField

This field is only a subclass of Django's `ImageField` with the ability to use images or thumbnails from another field.

This field came out of the need to offer a way to *optionally* override another field's thumbnail. To fall back to another field or thumbnail's value, simply define a dotted path to another model attribute. Using the example, above:

```
related_content_image = ImageFallbackField(
    fallback_path='artwork.thumbnails.related_content',
    upload_to='artwork/'
)
```

Since this is at its core an `ImageField`, using in a template is as simple as:

```
{# assuming 'object' is an instance of your model #}
{{ object.related_content_image.url }}
```

But, here's what makes it special – as defined, this field will do one of two things when accessed:

1. If the field **has a value** (a file has been uploaded), it will return that value.
2. If the field is **empty**, it will return the thumbnail from its fallback path, `artwork.thumbnails.related_content`.

## 4.3 Renderers

### 4.3.1 List of available renderers

#### ResizeRenderer

Resizes an image, maintaining the image's aspect ratio.

Example:

```
# resize an image to 150px wide, 84px tall
ResizeRenderer(150, 84)
```

If you would like to ignore the aspect ratio, pass in `constrain=False`.

#### CropRenderer

Resizes an image, crops to an area.

Example:

```
# resize an image to 150 on it's largest side,
# and select a 150x150 square from the center.x
CropRenderer(150, 150)
```

Internally, `CropRenderer` uses PIL's `ImageOps.fit` to scale and select the image.

#### LetterboxRenderer

Resizes an image using the *ResizeRenderer* and centers over a given background color. Background colors are provided as hex values.

Example:

```
# resize an image, place on black background
LetterboxRenderer(150, 150, bg_color='#000000')
```

### 4.3.2 Creating your own renderers

Better documentation forthcoming. In the meantime, subclass `undermythumb.renderers.BaseRenderer` and implement custom image logic in a method called `_render`.