Django-deepzoom Documentation

Release 3.0.3

David J Cox

May 25, 2015

Contents

1 What is Django-deepzoom? 3

2 How do I install it? 5

3 How do I configure it? 9

4 How is it licensed? 13

5 How can I leave feedback? 15

6 Search 17

Contents:

- *Django-deepzoom documentation*
  - *What is Django-deepzoom?*
  - *How do I install it?*
  - *How do I configure it?*
  - *How is it licensed?*
  - *How can I leave feedback?*
- *Search*

CHAPTER 1

What is Django-deepzoom?

Django-deepzoom is a drop-in Django app for the creation and use of Deep Zoom tiled images. It handily integrates Daniel Gasienica's and Kapil Thangavelu's deepzoom.py image generator and the OpenSeadragon deep zoom viewer into a set of model classes and template tags which programmatically generate tiled images and all JavaScript necessary for their instantiation into templates.

Django-deepzoom 3.0 involves major architectural changes. It introduces signal-based save, a new *DEFAULT_CREATE_DEEPZOOM_OPTION* setting, better file management, and decoupled file locations. It is Python 2/3 compatible, Django 1.4+ compatible, and Pillow 1.7.8+ compatible.

The purpose of Django-deepzoom is to make the integration of the deepzoom tiled image viewer into Django projects as easy as possible. Previously that required importing the standalone deepzoom module into your project, writing custom model class code to generate tiled images from it, and crafting custom JavaScript markup in your templates to instantiate it. Yikes.

Django-deepzoom handles this all for you. The app consists of two model classes and a template tag that orchestrate image uploads, deepzoom generation, deepzoom file saves/deletes to the filesystem, and template JavaScript markup.

The classes consist of an abstract image upload class and a standard deepzoom generator class.

The image upload class is just as it's named: a class that handles an image upload, names it, and generates a slug for it. However, it also provides a checkbox in the admin to request the generation of a deep zoom image from the uploaded image. Because the class is abstract and generates deep zooms conditionally, it can be inherited and extended for use by any class that incorporates uploaded images.

UploadedImage class:

- abstract class
- ImageField for uploaded image files
- saves unique image name
- saves unique image slug from image name
- saves image height, width
- configurable image file save location
- optional deep zoom generation
- image file deletion on delete

The standard deepzoom class handles deepzoom generation and file management. It receives a reference to the uploaded image from the image upload class, generates a deep zoom image from it, names and slugifies it identically to the uploaded image, extracts the XML signature needed to instantiate the deepzoom viewer, saves the XML needed to instantiate the deepzoom viewer, and saves the generated file hierarchy to the filesystem. It does not allow editing of

existing deepzooms, but it does allow for deletion and re-generation, because it handles file hierarchy deletion from the filesystem on delete.

DeepZoom class:

- standard class
- generates deep zoom tiled image
- saves unique deepzoom image name (named identically to associated image)
- saves unique deepzoom image slug (slugified identically to associated image)
- saves file path of associated image
- saves deepzoom image path (part after MEDIA_ROOT)
- saves full deepzoom image path
- saves dzi image file path needed by deep zoom viewer

The django-deepzoom template tag takes a deepzoom object and a div id and outputs the full JavaScript needed to instantiate the specified deepzoom object in the deepzoom viewer within the specified div and enable touch gestures for it. Since it's a custom template tag, it needs to be loaded somewhere before its usage in the template by calling {% load deepzoom_tags %}. Since it outputs JavaScript only and not HTML, the deepzoom template tag needs to be embedded within <script> tags.

DeepZoom template tag usage:

```
{% load deepzoom_tags %}
<div id="deepzoom_div"></div>
<script>
{% deepzoom_js deepzoom_obj "deepzoom_div" %}
</script>
```

Neither the deep zoom queryset object nor the deep zoom div ID have to be named like in the example. Any name can be given to either, so long as the deep zoom queryset object is named the same way in the template as it is in the view providing it and the deep zoom div ID matches the name is passed deepzoom_js template tag. Further, the deep zoom div is just the container for the deep zoom viewer, so it can be used any way that divs can, including as a floated element, a modal dialog, etc. Knock yourself out.

CHAPTER 2

How do I install it?

1.) Install "django-deepzoom" like this:

```
pip install -U django-deepzoom
```

or, like this:

```
wget https://pypi.python.org/packages/source/d/django-deepzoom/django-deepzoom-<VERSION>.tar.gz
tar -xvf django-deepzoom-<VERSION>.tar.gz
cd django-deepzoom-<VERSION>
python setup.py install
```

If you download it directly, change <VERSION> to one of the available version numbers, of course. Installing to a virtualenv is a good idea, too.

2.) Add "deepzoom" to your INSTALLED_APPS setting. Django 1.7 introduced the *AppConfig.ready()* entry point for app intialization which is needed for the new signals design (in that version of Django). That means the django-deepzoom app needs to be specified one way in Django 1.7+ and the traditional way in previous Django versions. In Django 1.7+ add the app like this:

```
(in settings.py)

INSTALLED_APPS = (
    ...
    'deepzoom.apps.DeepZoomAppConfig',
    ...
)
```

However, in Django 1.6 and before, add the app the traditional way, like this:

```
(in settings.py)

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

3.) Sub-class the '*UploadedImage*' model class as your own (image-based) class, something like this:

```
(in models.py)

from deepzoom.models import DeepZoom, UploadedImage

class MyImage(UploadedImage):
```

```
  '''
  Overrides UploadedImage base class.
  '''
  pass
```

The save() method of the overridden class can be overridden, too, of course, to add additional fields or features.

4.) Run *python manage.py syncdb* to create the django-deepzoom models.

5.) Add an appropriate URL to your Urlconf, something like this:

```
(in urls.py)

from deepzoom.views import deepzoom_view

urlpatterns = patterns('',
    ...
    url(r'^deepzoom/(?P<passed_slug>\b[a-z0-9\-]+\b)',
        deepzoom_view,
        name="v_deepzoom"),
    ...
)
```

The slug parameter name does not have to be the same as the example as long as it matches the corresponding view. (See below)

6.) Write a view that queries for a specific DeepZoom object and passes it to a template, something like this:

```
(in views.py)

from deepzoom.models import DeepZoom

def deepzoom_view(request, passed_slug=None):
  try:
      _deepzoom_obj = DeepZoom.objects.get(slug=passed_slug)
  except DeepZoom.DoesNotExist:
      raise Http404
  return render_to_response('deepzoom.html',
                            {'deepzoom_obj': _deepzoom_obj},
                            context_instance=RequestContext(request))
```

The slug parameter name does not have to be the same as the example as long as it matches the corresponding urlconf signature. (See above)

7.) In your template, create an empty div with a unique ID. Load the deepzoom tags and pass the deepzoom object and deepzoom div ID to the template tag inside a <script> block in the body like this:

```
(in e.g. deepzoom.html)

{% extends "base.html" %}

{% load deepzoom_tags %}

<div id="deepzoom_div"></div>

<script>{% deepzoom_js deepzoom_obj "deepzoom_div" %}</script>
```

Neither the deep zoom queryset object nor the deep zoom div ID have to be named like in the example. Any name can be given to either, so long as the deep zoom queryset object name used in the template matches the queryset object

name used in the view providing it and the deep zoom div ID matches the name passed to the deepzoom_js template tag.

8.) Run *python manage.py collectstatic* to collect your static files into STATIC_ROOT, specifically so that the openseadragon files are available.

9.) Start the development server and visit *http://127.0.0.1:8000/admin/* to upload an image to the associated model (you'll need the Admin app enabled). Be sure to check the *Generate deep zoom?* checkbox for that image before saving it.

CHAPTER 3

How do I configure it?

Django-deepzoom does not require configuration to work. It assumes a sensible default configuration that should work for anyone. However, should you wish to customize its behavior, the following can be configured in your settings.py file.

UPLOADEDIMAGE_ROOT

A string defining the directory to be appended to MEDIA_ROOT for storing uploaded images. Do not include beginning or trailing directory separators. If defined, but not actually created, your directory will be created for you. If left undefined, 'uploaded_images' is used as the default directory name.

E.g. after prepending your media root, the default UPLOADEDIMAGE_ROOT is *'/path/to/media_root/uploaded_images/'*.

Or, if you define `UPLOADEDIMAGE_ROOT='my/uploaded/images'`, the final path will be *'/path/to/media_root/my/uploaded/images/'*.

DEEPZOOM_PARAMS

This is a dictionary of arguments used to initialize the deep zoom creator, including 'tile_size', 'tile_overlap', 'tile_format', 'image_quality', and 'resize_filter'. If undefined, `{'tile_size': 256, 'tile_overlap': 1, 'tile_format': "jpg", 'image_quality': 0.85, 'resize_filter': "antialias"}` is used by default.

tile_size

- type: int
- options: 1 to maxint
- default: 256

The tile_size defines the size of tiles that each image on a pyramid level will be subdivided into and resized for the next pyramid level. The smaller the size, the deeper the overall zoom. However, as the tile size approaches one, the number of tiles per level increases to the limit of pixels of the largest side of the original image overtaxing file IO and server bandwidth per page request. Conversely, as the tile size approaches the number of pixels of the largest side of the original image, the number of tiles approaches one, eliminating the zoom effect completely. The most useful range is roughly 1/10 - 1/4 the size of the largest side of the original image.

tile_overlap

- type: int
- options: 0 to 10
- default: 1

The tile_overlap defines the number of pixels a tile overlaps its neighboring tiles on each pyramid level. As tile overlap is decreased, the total number of tiles generated potentially reduces, and as tile overlap is increased, the total number of tiles generated potentially increases. If set to zero, gapping between tiles may be visible when zoomed.

tile_format

- type: str
- options: 'jpg' or 'png'
- default: 'jpg'

The tile_format determines the final image type of the generated tiled images. The usual image format considerations apply. Generally, JPEG is better suited for photographs and realistic images, whereas PNG is better suited for line drawings or text. If file size and bandwidth is not a concern, PNG will be superior because it is a lossless compression format.

image_quality

- type: float
- options: 0.00 to 1.00
- default: 0.85

The image_quality setting pertains only if tile_format is set to 'jpg', because it is a JPEG setting. It will not influence anything if tile_format is set to 'png' because PNGs do not care about it. (JPEGs are so special...) It specifies the quality of JPEG image conversion. Lower settings produce grainier images of smaller file size while higher settings produce sharper images of larger size. General consensus is that the average viewer is unable to detect quality improvements at settings above 0.80 (80%). However, since this is a re-sampling of an already-compressed image in most circumstances, it seemed prudent to set the default higher. Experiment.

resize_filter

- type: str
- options: 'cubic', 'bilinear', 'bicubic', 'nearest', or 'antialias'
- default: 'antialias'

The resize_filter is the method used to re-sample images when resizing them during tile creation. Different filters are better suited for certain tasks. The 'antialias' filter trades off highest quality for slowest speed of creation. Since tiled image generation is a one-time expense, it's a reasonable tradeoff for the default.

DEEPZOOM_ROOT

A string defining the directory to be appended to MEDIA_ROOT for storing deep zoom files. Do not include beginning or trailing directory separators. If defined, but not actually created, your directory will be created for you. If left undefined, 'deepzoom_images' is used as the default directory name.

E.g. after prepending your media root, the default DEEPZOOM_ROOT is *'/path/to/media_root/deepzoom_images/'*.

Or, if you define `DEEPZOOM_ROOT='my/deepzoom/images'`, the final path will be *'/path/to/media_root/my/deepzoom/images/'*.

DEFAULT_CREATE_DEEPZOOM_OPTION

A Boolean value that sets the default value of the *create_deepzoom* field globally. By setting the *DEFAULT_CREATE_DEEPZOOM_OPTION* to *True* or *False*, new instances of a *UploadedImage* subclass will be set to always create a deepzoom or never to create a deepzoom.

LOGGING

Certain non-critical exceptions are logged instead of thrown. To capture the log messages, add this logging configuration to your settings.py file:

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'file': {
            'level': 'ERROR',
            'class': 'logging.FileHandler',
            'filename': 'deepzoom.exception.log',
        },
    },
    'loggers': {
        'deepzoom.models': {
            'handlers': ['file'],
            'level': 'ERROR',
            'propagate': True,
        },
    },
}
```

CHAPTER 4

How is it licensed?

Django-deepzoom is BSD-licensed for full, unfettered use as long as attribution is given to the author.

CHAPTER 5

How can I leave feedback?

Send questions, suggestions, comments to <davidjcox.at@gmail.com>.

Let me know if you're successfully using Django-deepzoom for your project.

Build good things.

CHAPTER 6

Search

- search