Django utils2 Documentation

Release 2.2.1

Rick van Hattem

February 13, 2017

Contents

Contents:

CHAPTER 1

Introduction

Travis status: Coverage: Django Utils is a collection of small Django helper functions and classes which make common patterns shorter and easier. It is by no means a complete collection but it has served me quite a bit in the past and I will keep extending it.

The library depends on the Python Utils library.

Documentation is available at: http://django-utils2.readthedocs.org/en/latest/

1.1 Install

To install:

1. Run *pip install django-utils2* or execute *python setup.py install* in the source directory
2. Add *django_utils* to your *INSTALLED_APPS*

If you want to run the tests, run *py.test* (requirements in *tests/requirements.txt*)

1.2 Links

- **Documentation**
  - http://django-utils2.readthedocs.org/en/latest/
- **Source**
  - https://github.com/WoLpH/django-utils
- **Bug reports**
  - https://github.com/WoLpH/django-utils/issues
- **Package homepage**
  - https://pypi.python.org/pypi/django-utils2
- **My blog**
  - http://w.wol.ph/

CHAPTER 2

django_utils Package

2.1 django_utils Package

2.2 base_models Module

2.3 choices Module

2.3.1 Usage

Create a *Choices* class and add *Choice* objects to the class to define your choices.

Example with explicit values:

The normal Django version:

```
class Human(models.Model):
    GENDER = (
        ('m', 'Male'),
        ('f', 'Female'),
        ('o', 'Other'),
    )
    gender = models.CharField(max_length=1, choices=GENDER)
```

The Django Utils Choices version:

```
from django_utils import choices

class Human(models.Model):
    class Gender(choices.Choices):
        Male = choices.Choice('m')
        Female = choices.Choice('f')
        Other = choices.Choice('o')

    gender = models.CharField(max_length=1, choices=Gender.choices)
```

To reference these properties:

```
Human.create(gender=Human.Gender.Male)
```

Example with implicit values:

The normal Django version:

```
class SomeModel(models.Model):
    SOME_ENUM = (
        (1, 'foo'),
        (2, 'bar'),
        (3, 'spam'),
        (4, 'eggs'),
    )
    enum = models.IntegerField(choices=SOME_ENUM, default=1)
```

The Django Utils Choices version:

```
from django_utils import choices

class SomeModel(models.Model):
    class Enum(choices.Choices):
        Foo = choices.Choice()
        Bar = choices.Choice()
        Spam = choices.Choice()
        Eggs = choices.Choice()

    enum = models.IntegerField(
        choices=Enum.choices, default=Enum.Foo)
```

To reference these properties:

```
SomeModel.create(enum=SomeModel.Enum.Spam)
```

class django_utils.choices.**Choice**(*value=None*, *label=None*)

Bases: object

The choice object has an optional label and value. If the value is not given an autoincrementing id (starting from 1) will be used

```
>>> choice = Choice('value', 'label')
>>> choice
<Choice[1]:label>
>>> str(choice)
'label'
```

```
>>> choice = Choice()
>>> choice
<Choice[2]:None>
>>> str(choice)
'None'
```

order = 0

class django_utils.choices.**Choices**

Bases: object

The choices class is what you should inherit in your Django models

```
>>> choices = Choices()
>>> choices.choices[0]
Traceback (most recent call last):
...
KeyError: 'Key 0 does not exist'
```

```
>>> choices.choices
OrderedDict()
>>> str(choices.choices)
'OrderedDict()'
>>> choices.choices.items()
[]
```

```
>>> class ChoiceTest(Choices):
...     a = Choice()
>>> choices = ChoiceTest()
>>> choices.choices.items()
[(0, <Choice[3]:a>)]
>>> choices.a
0
>>> choices.choices['a']
<Choice[3]:a>
>>> choices.choices[0]
<Choice[3]:a>
```

choices = OrderedDict()

class django_utils.choices.**ChoicesDict**
Bases: object

The choices dict is an object that stores a sorted representation of the values by key and database value

items()

class django_utils.choices.**ChoicesMeta**
Bases: type

The choices metaclass is where all the magic happens, this automatically creates a ChoicesDict to get a sorted list of keys and values

2.4 queryset Module

django_utils.queryset.**queryset_iterator**(*queryset*, *chunksize=1000*, *getfunc=<built-in function getattr>*)
'' Iterate over a Django Queryset ordered by the primary key

This method loads a maximum of chunksize (default: 1000) rows in it's memory at the same time while django normally would load all rows in it's memory. Using the iterator() method only causes it to not preload all the classes.

Note that the implementation of the iterator does not support ordered query sets.

2.5 `view_decorators` Module

2.6 `utils` Module

2.7 Subpackages

2.7.1 django_utils.management module

Subpackages

django_utils.management.commands package

Submodules

commands.admin_autogen module

commands.base_command module

commands.settings module

Module contents

Module contents

2.7.2 django_utils.templatetags package

Submodules

django_utils.templatetags.debug module

Module contents

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

d

C

D

I

O

Q