Django-Choices Documentation

Release 1.6.0

Jason Webb, Sergei Maertens

Jul 13, 2017

Contents

Order and sanity for django model choices.

Contents:

CHAPTER 1

Choice items

The `ChoiceItem` class is what drives the choices. Each instance corresponds to a possible choice for your field.

Basic usage

```
class MyChoices(DjangoChoices):
    my_choice = ChoiceItem(1, 'label 1')
```

The first argument for `ChoiceItem` is the value, as it will be stored in the database. `ChoiceItem` values can be any type, as long as it matches the field where the choices are defined, e.g.:

String type:

```
class Strings(DjangoChoices):
    one = ChoiceItem('one', 'one')


class Model1(models.Model):
    field = models.CharField(max_length=10, choices=Strings.choices)
```

or integer:

```
class Ints(DjangoChoices):
    one = ChoiceItem(1, 'one')


class Model2(models.Model):
    field = models.IntegerField(choices=Ints.choices)
```

There is also a 'short name'. You can import `C` instead of *ChoiceItem*, if you're into that.

Custom attributes

Any additional (custom) keyword arguments passed to the constructor, are made available as custom attributes:

```
>>> choice = ChoiceItem('excellent', limit_to=['US', 'CA', 'CN'])
>>> choice.limit_to
['US', 'CA', "CN"]
```

To obtain the `ChoiceItem` instance, see *get_choice*

Labels

The second argument to the *ChoiceItem* class is the label. It's recommended to specify this explicitly if you use internationalization, e.g.:

```
from django.utils.translation import ugettext_lazy as _


class MyChoices(DjangoChoices):
    one = ChoiceItem(1, _('one'))
```

If the label is not provided, it will be automatically determined from the class property, and underscores are translated to spaces. So, the following example yields:

```
>>> class MyChoices(DjangoChoices):
...     first_choice = ChoiceItem(1)

>>> MyChoices.choices
((1, 'first choice'),)
```

Ordering

ChoiceItem objects also support ordering. If not provided, the choices are returned in order of declaration.

```
>>> class MyChoices(DjangoChoices):
...     first = ChoiceItem(1, order=20)
...     second = ChoiceItem(2, order=10)

>>> MyChoices.choices
(
    (2, 'second'),
    (1, 'first'),
)
```

Values

If you really want to use the minimal amount of code, you can leave off the value as well, and it will be determined from the label.

```
>>> class Sample(DjangoChoices):
...     OptionA = ChoiceItem()
...     OptionB = ChoiceItem()

>>> Sample.choices
(
    ('OptionA', 'OptionA'),
    ('OptionB', 'OptionB'),
)
```

`DjangoChoices` class attributes

The choices class itself has a few useful attributes. Most notably *choices*, which returns the choices as a tuple.

choices

```
>>> class Sample(DjangoChoices):
...     OptionA = ChoiceItem()
...     OptionB = ChoiceItem()

>>> Sample.choices
(
    ('OptionA', 'OptionA'),
    ('OptionB', 'OptionB'),
)
```

labels

Returns a dictionary with a mapping from label to value:

```
>>> class MyChoices(DjangoChoices):
...     first_choice = ChoiceItem(1)
...     second_choice = ChoiceItem(2)

>>> MyChoices.labels
{'first_choice': 1, 'second_choice': 2}
```

values

Returns a dictionary with a mapping from value to label:

```
>>> class MyChoices(DjangoChoices):
...     first_choice = ChoiceItem(1, 'label 1')
...     second_choice = ChoiceItem(2, 'label 2')

>>> MyChoices.values
{1: 'label 1', '2': 'label 2'}
```

validator

Note: At least since Django 1.3, there is model and form-level validation of the choices. Unless you have a reason to explicitly specify/override the validator, you can skip specifying this validator.

Returns a validator that can be used in your model field. This validator checks that the value passed to the field is indeed a value specified in your choices class.

attributes

Returns an `OrderedDict` with the mapping from choice value -> attribute on the choices class.

```
>>> class MyChoices(DjangoChoices):
...     first_choice = ChoiceItem(1, 'label 1')
...     second_choice = ChoiceItem(2, 'label 2')

>>> MyChoices.attributes
OrderedDict([(1, 'first_choice'), (2, 'second_choice')])
```

get_choice

Returns the actual `ChoiceItem` instance for a given value:

```
>>> class MyChoices(DjangoChoices):
...     first_choice = ChoiceItem(1, 'label 1')
...     second_choice = ChoiceItem(2, 'label 2')

>>> MyChoices.get_choice(MyChoices.second_choice)
<ChoiceItem value=2 label='label 2' order=1>
```

This allows you to inspect any `ChoiceItem` attributes.

CHAPTER 2

Contributing

To get up and running quickly, fork the github repository and make all your changes in your local clone.

Git-flow is prefered as git workflow, but as long as you make pull requests against the *develop* branch, all should be well. Pull requests should always have tests, and if relevant, documentation updates.

Feel free to create unfinished pull-requests to get the tests to build and get work going, someone else might always want to pick up the tests and/or documentation.

Testing

For testing the standard unittest2 library is used, contrary to Django's test framework. The reason is simple: speed. Django choices doesn't touch the database in any way, so the standard library unit tests are fine.

To run the tests in your (virtual) environment, simple execute

```
python runtests.py
```

This will run the tests with the current python version and Django version.

To run the tests on all supported python/Django versions, use tox.

```
pip install tox
tox
```

If you want to speed this up, you can also use detox. This library will run as much in parallel as possible.

Documentation

The documentation is built with Sphinx. Run *make* to build the documentation:

You can now open *_build/index.html*.

Coding style

Please stick to PEP8, and use pylint or similar tools to check the code style.

CHAPTER 3

Overview

Django choices provides a declarative way of using the choices option on django fields.

CHAPTER 4

Requirements

Django choices is fairly simple, so most Python and Django versions should work. It is tested against Python 2.7, 3.3, 3.4, 3.5 and PyPy. Django 1.8 until and including 1.11 alpha are supported (and tested in Travis).

If you need to support older Python or Django versions, you should stick with version `1.4.4`. Backwards compatibility is dropped from 1.5 onwards.

CHAPTER 5

Quick-start

Install like any other library:

```
pip install django-choices
```

There is no need to add it in your installed apps.

To use it, you write a choices class, and use it in your model fields:

```
from djchoices import ChoiceItem, DjangoChoices


class Book(models.Model):

    class BookType(DjangoChoices):
        short_story = ChoiceItem('short', 'Short story')
        novel = ChoiceItem('novel', 'Novel')
        non_fiction = ChoiceItem('non_fiction', 'Non fiction')


    author = models.ForeignKey('Author')
    book_type = models.CharField(
        max_length=20, choices=BookType.choices,
        default=BookType.novel
    )
```

You can then use the availe choices in other modules, e.g.:

```
from .models import Book

Person.objects.create(author=my_author, type=Book.BookTypes.short_story)
```

The `DjangoChoices` classes can be located anywhere you want, for example you can put them outside of the model declaration if you have a 'common' set of choices for different models. Any place is valid though, you can group them all together in `choices.py` if you want.

CHAPTER 6

License

Licensed under the MIT License.

CHAPTER 7

Souce Code and contributing

The source code can be found on github.

Bugs can also be reported on the github repository, and pull requests are welcome. See *Contributing* for more details.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search