Django EnumChoiceField Documentation

Release 2.0.0

Tim Heap

Dec 30, 2019

CONTENTS

1 Setup 3

2 Usage 5

3 EnumChoiceField 7

4 Enum classes 9

5 ORM Queries 11

6 Using with the Django admin 13

Python Module Index 15

Index 17

For a quick example, check out the code below:

```
from enumchoicefield import ChoiceEnum, EnumChoiceField

class Fruit(ChoiceEnum):
    apple = "Apple"
    banana = "Banana"
    orange = "Orange"

class Profile(models.Model):
    name = models.CharField(max_length=100)
    favourite_fruit = EnumChoiceField(Fruit, default=Fruit.banana)


citrus_lovers = Profile.objects.filter(favourite_fruit=Fruit.orange)
```

Contents:

CHAPTER

ONE

SETUP

`django-enumchoicefield` is compatible with Django 1.8 and higher, Python 2.7, and Python 3.4 and higher.

You can install `django-enumchoicefield` using pip:

```
$ pip install django-enumchoicefield
```

On Python 2.7, you will also need to install `enum34`:

```
$ pip install enum34
```

CHAPTER

TWO

USAGE

The following code outlines the most simple usecase of `EnumChoiceField`:

```
from enumchoicefield import ChoiceEnum, EnumChoiceField

class Fruit(ChoiceEnum):
    apple = "Apple"
    banana = "Banana"
    orange = "Orange"

class Profile(models.Model):
    name = models.CharField(max_length=100)
    favourite_fruit = EnumChoiceField(Fruit, default=Fruit.banana)


citrus_lovers = Profile.objects.filter(favourite_fruit=Fruit.orange)
```

The enumerations should extend the *`ChoiceEnum`* class. For each member in the enumeration, their human-readable name should be their value. This human-readable name will be used when presenting forms to the user.

For more advanced usage, refer to the documentation on *EnumChoiceField*, *Enum classes*, or *ORM Queries*.

CHAPTER

THREE

ENUMCHOICEFIELD

class enumchoicefield.fields.**EnumChoiceField**(*enum_class*, ...)

Create an EnumChoiceField. This field generates choices from an `enum.Enum`.

The *`EnumChoiceField`* extends `django.db.models.Field`. It accepts one additional argument: `enum_class`, which should be a subclass of `enum.Enum`. It is recommended that this enum subclasses *`ChoiceEnum`*, but this is not required.

When saving enum members to the database, The chosen member is stored in the database using its `name` attribute. This keeps the database representation stable when adding and removing enum members.

A `max_length` is automatically generated from the longest `name`. If you add a new enum member with a longer name, or remove the longest member, the generated `max_length` will change. To prevent this, you can manually set a `max_length` argument, and this will be used instead.

If a default choice is supplied, the enum class must have a `deconstruct` method. If the enum inherits from *`DeconstructableEnum`*, this will be handled for you.

The display value for the Enums is taken from the `str` representation of each value. By default this is something like `MyEnum.foo`, which is not very user friendly. *`PrettyEnum`* makes defining a human-readable `str` representation easy.

CHAPTER

FOUR

ENUM CLASSES

class enumchoicefield.enum.**PrettyEnum**

A *PrettyEnum* makes defining nice, human-readable names for enum members easy. To use it, subclass *PrettyEnum* and declare the enum members with their human-readable name as their value:

```
class Fruit(PrettyEnum):
    apple = "Apple"
    banana = "Banana"
    orange = "Orange"
```

The members' values will be automatically set to ascending integers, starting at one. In the example above, `Fruit.apple.value` is `1`, and `Fruit.orange.value` is `3`.

class enumchoicefield.enum.**DeconstructableEnum**

deconstruct()

a *DeconstructableEnum* defines *deconstruct()*, compatible with Django migrations. If you want to set a default for an *EnumChoiceField*, the enum must be deconstructable.

class enumchoicefield.enum.**ChoiceEnum**

a *ChoiceEnum* extends both *PrettyEnum* and *DeconstructableEnum*. It is recommended to use a *ChoiceEnum* subclass with *EnumChoiceField*, but this is not required.

CHAPTER

FIVE

ORM QUERIES

You can filter and search for enum members using standard Django ORM queries. The following queries demonstrate some of what is possible:

```
from enumchoicefield import ChoiceEnum, EnumChoiceField

class Fruit(ChoiceEnum):
    apple = "Apple"
    banana = "Banana"
    lemon = "Lemon"
    lime = "Lime"
    orange = "Orange"

class Profile(models.Model):
    name = models.CharField(max_length=100)
    favourite_fruit = EnumChoiceField(Fruit, default=Fruit.banana)


apple_lovers = Profile.objects.filter(favourite_fruit=Fruit.apple)
banana_haters = Profile.objects.exclude(favourite_fruit=Fruit.banana)

citrus_fans = Profile.objects.filter(
    favourite_fruit__in=[Fruit.orange, Fruit.lemon, Fruit.lime])
```

5.1 Ordering

Ordering on a *EnumChoiceField* field will order results alphabetically by the `names` of the enum members, which is probably not useful. To order results by an enum value, *enumchoicefield.utils.order_enum()* can be used.

`enumchoicefield.utils.`**`order_enum`**(*field*, *members*)

Make an annotation value that can be used to sort by an enum field.

`field` The name of an EnumChoiceField.

`members` An iterable of Enum members in the order to sort by.

Use like:

```
desired_order = [MyEnum.bar, MyEnum.baz, MyEnum.foo]
ChoiceModel.objects\
    .annotate(my_order=order_enum('choice', desired_order))\
    .order_by('my_order')
```

As Enums are iterable, `members` can be the Enum itself if the default ordering is desired:

```
ChoiceModel.objects\
    .annotate(my_order=order_enum('choice', MyEnum))\
    .order_by('my_order')
```

> **Warning:** On Python 2, Enums may not have a consistent order, depending upon how they were defined. You can set an explicit order using `__order__` to fix this. See the `enum34` docs for more information.

Any enum members not present in the list of members will be sorted to the end of the results.

5.2 Undefined behaviour

Internally, the enum member is stored as a CharField using the `name` attribute. Any operation that CharFields support are also supported by an *`EnumChoiceField`*. Not all of these operations make sense, such as `contains`, `gt`, and `startswith`, and may not behave in a sensible manner.

CHAPTER

SIX

USING WITH THE DJANGO ADMIN

*EnumChoiceField*s are compatible with the Django admin out of the box, with one exception. If you want to use a *EnumChoiceField* in a `list_filter`, you need to use the *EnumListFilter*.

class `enumchoicefield.admin.`**`EnumListFilter`**(**args*, ***kwargs*)

A FieldListFilter for use in Django admin in combination with an *EnumChoiceField*. Use like:

```
class FooModelAdmin(ModelAdmin):
    list_filter = [
        ('enum_field', EnumListFilter),
    ]
```

PYTHON MODULE INDEX

e

enumchoicefield.enum, 9
enumchoicefield.fields, 7
enumchoicefield.utils, 11

INDEX

C

`ChoiceEnum` (*class in enumchoicefield.enum*), 9

D

`deconstruct()` (*enumchoicefield.enum.DeconstructableEnum method*), 9

`DeconstructableEnum` (*class in enumchoicefield.enum*), 9

E

`EnumChoiceField` (*class in enumchoicefield.fields*), 7

`enumchoicefield.enum` (*module*), 9

`enumchoicefield.fields` (*module*), 7

`enumchoicefield.utils` (*module*), 11

`EnumListFilter` (*class in enumchoicefield.admin*), 13

O

`order_enum()` (*in module enumchoicefield.utils*), 11

P

`PrettyEnum` (*class in enumchoicefield.enum*), 9