django-composite-field

Release 1.0.0

Jan 06, 2020

Contents

This is an implementation of a CompositeField for Django. Composite fields can be used to group fields together and reuse their definitions.

CHAPTER 1

Installation instructions

Thanks for downloading django-composite-field.

To install it, run the following command inside this directory:

```
python setup.py install
```

If you have the Python `pip` utility available, you can type the following to download and install in one step:

```
pip install django-composite-field
```

Or if you're using `pipenv`:

```
pipenv install django-composite-field
```

Or if you'd prefer you can simply place the included `composite_field` directory somewhere on your Python path, or symlink to it from somewhere on your Python path; this is useful if you're working from a Git checkout.

Note that this application requires Python 3.4 or later, and a functional installation of Django 1.8 or newer. You can obtain Python from https://www.python.org/ and Django from https://www.djangoproject.com/.

CHAPTER 2

Usage

2.1 Example

```
class CoordField(CompositeField):
    x = models.FloatField()
    y = models.FloatField()

class Place(models.Model):
    name = models.CharField(max_length=10)
    coord = CoordField()

p = Place(name='Foo', coord_x=42, coord_y=0)
q = Place(name='Foo', coord=p.coord)
q.coord.y = 42
```

2.2 How does it work?

The content of composite fields are stored inside the model, so they do not have to fiddle with any internals of the Django models. In the example above `p.coord` returns a proxy object that maps the fields `x` and `y` to the model fields `coord_x` and `coord_y`.

This is roughly equivalent to the following code:

```
class CoordProxy:

    def __init__(self, model):
        self.model = model

    def __get__(self, instance):
        return CoordProxy(instance)
```

(continues on next page)

(continued from previous page)

```
    def __set__(self, instance, value):
        instance.coord_x = value.coord_x
        instance.coord_y = value.coord_y

    @property
    def x(self):
        return self.model.coord_x

    @x.setter
    def x(self, value):
        self.model.coord_x = value

    @property
    def y(self):
        return self.model.coord_y

    @y.setter
    def y(self, value):
        self.model.coord_y = value

class Place(models.Model):
    name = models.CharField(max_length=10)
    coord_x = models.FloatField()
    coord_y = models.FloatField()
    coord = CoordProxy()
```

2.3 Advanced usage

The proxy object also makes it possible to assign more than one property at once:

```
place1.coord = place2.coord
```

It also supports using dictionaries for the `__init__` method or assigning them as a value:

```
place1 = Place(coord={'x': 42, 'y': 0})
place1.coord = {'x': 43, 'y': 1}
```

It is even possible to replace the `Proxy` object entirely and return a custom type. A good example for this is the included `ComplexField` which stores a `complex` number in two integer fields.

CHAPTER 3

Examples

The following code is part of the unit tests and should be mostly self explaining.

3.1 models.py

```
from django.db import models

from composite_field import CompositeField
from composite_field import LocalizedCharField
from composite_field import ComplexField


class CoordField(CompositeField):
    x = models.FloatField(null=True)
    y = models.FloatField(null=True)

    class Proxy(CompositeField.Proxy):

        def __bool__(self):
            return self.x is not None and self.y is not None


class Place(models.Model):
    name = models.CharField(max_length=10)
    coord = CoordField()


class PlaceWithDefaultCoord(models.Model):
    name = models.CharField(max_length=10)
    coord = CoordField(default={'x': 1.0, 'y': 2.0})


class Direction(models.Model):
```

(continues on next page)

(continued from previous page)

```
    source = CoordField()
    distance = models.FloatField()
    target = CoordField()


class LocalizedFoo(models.Model):
    id = models.AutoField(primary_key=True)
    name = LocalizedCharField(languages=('de', 'en'), max_length=50)

    def __str__(self):
        return self.name.current


class ComplexTuple(models.Model):
    x = ComplexField(blank=True, null=True)
    y = ComplexField(blank=False, null=False, verbose_name='Y')
    z = ComplexField(verbose_name='gamma')


class ComplexTupleWithDefaults(models.Model):
    x = ComplexField(blank=True, null=True, default=None)
    y = ComplexField(blank=False, null=False, default=42)
    z = ComplexField(default=42j)


class TranslatedAbstractBase(models.Model):
    name = LocalizedCharField(languages=('de', 'en'), max_length=50)

    class Meta:
        abstract = True


class TranslatedModelA(TranslatedAbstractBase):
    pass


class TranslatedModelB(TranslatedAbstractBase):
    pass


class TranslatedNonAbstractBase(models.Model):
    name = LocalizedCharField(languages=('de', 'en'), max_length=50)

    class Meta:
        abstract = False


class TranslatedModelC(TranslatedNonAbstractBase):
    pass


class TranslatedModelD(TranslatedNonAbstractBase):
    pass
```

3.2 tests.py

```
import math
import unittest

import django
import django.test
from django.test import TestCase
try:
    from django.urls import reverse
except ImportError:
    from django.core.urlresolvers import reverse
from django.utils import translation
from django.utils.encoding import force_text

from composite_field_test.models import Place
from composite_field_test.models import PlaceWithDefaultCoord
from composite_field_test.models import Direction
from composite_field_test.models import LocalizedFoo
from composite_field_test.models import ComplexTuple
from composite_field_test.models import ComplexTupleWithDefaults
from composite_field_test.models import TranslatedAbstractBase
from composite_field_test.models import TranslatedModelA
from composite_field_test.models import TranslatedModelB
from composite_field_test.models import TranslatedNonAbstractBase
from composite_field_test.models import TranslatedModelC
from composite_field_test.models import TranslatedModelD


class CompositeFieldTestCase(TestCase):

    def test_repr(self):
        place = Place(coord_x=12.0, coord_y=42.0)
        self.assertEqual(repr(place.coord), 'CoordField(x=12.0, y=42.0)')

    def test_cmp(self):
        place1 = Place(coord_x=12.0, coord_y=42.0)
        place2 = Place(coord_x=42.0, coord_y=12.0)
        self.assertNotEqual(place1.coord, place2.coord)
        place2.coord.x = 12.0
        place2.coord.y = 42.0
        self.assertEqual(place1.coord, place2.coord)

    def test_assign(self):
        place1 = Place(coord_x=12.0, coord_y=42.0)
        place2 = Place()
        place2.coord = place1.coord
        self.assertEqual(place1.coord, place2.coord)
        place2 = Place(coord=place1.coord)
        self.assertEqual(place1.coord, place2.coord)

    def test_setattr(self):
        place = Place()
        place.coord.x = 12.0
        place.coord.y = 42.0
        self.assertEqual(place.coord_x, 12.0)
        self.assertEqual(place.coord_y, 42.0)
```

(continues on next page)

(continued from previous page)

```
        self.assertEqual(place.coord.x, 12.0)
        self.assertEqual(place.coord.y, 42.0)

    def test_field_order(self):
        fields = Place._meta.fields
        get_field = Place._meta.get_field
        name = get_field('name')
        coord_x = get_field('coord_x')
        coord_y = get_field('coord_y')
        self.assertTrue(fields.index(name) < fields.index(coord_x))
        self.assertTrue(fields.index(coord_x) < fields.index(coord_y))

    def test_field_order2(self):
        fields = Direction._meta.fields
        get_field = Direction._meta.get_field
        source_x = get_field('source_x')
        source_y = get_field('source_y')
        distance = get_field('distance')
        target_x = get_field('target_x')
        target_y = get_field('target_y')
        self.assertTrue(fields.index(source_x) < fields.index(source_y))
        self.assertTrue(fields.index(source_y) < fields.index(distance))
        self.assertTrue(fields.index(distance) < fields.index(target_x))
        self.assertTrue(fields.index(target_x) < fields.index(target_y))

    def test_modelform(self):
        from django import forms
        class DirectionForm(forms.ModelForm):
            class Meta:
                model = Direction
                exclude = ()
        form = DirectionForm()
        form = DirectionForm({})
        form.is_valid()

    def test_modelform_with_exclude(self):
        from django import forms
        class LocalizedFooForm(forms.ModelForm):
            class Meta:
                model = LocalizedFoo
                exclude = ()
        form = LocalizedFooForm()
        form = LocalizedFooForm({})
        self.assertFalse(form.is_valid())
        form = LocalizedFooForm({'name_de': 'Banane', 'name_en': 'Banana'})
        self.assertTrue(form.is_valid())
        foo = form.save(commit=False)
        self.assertEquals(foo.name.de, 'Banane')
        self.assertEquals(foo.name.en, 'Banana')

    def test_modelform_with_fields(self):
        from django import forms
        class LocalizedFooForm(forms.ModelForm):
            class Meta:
                model = LocalizedFoo
                fields = ('name_de', 'name_en')
        form = LocalizedFooForm()
```

(continues on next page)

(continued from previous page)

```
        form = LocalizedFooForm({})
        self.assertFalse(form.is_valid())
        form = LocalizedFooForm({'name_de': 'Banane', 'name_en': 'Banana'})
        self.assertTrue(form.is_valid())
        foo = form.save(commit=False)
        self.assertEquals(foo.name.de, 'Banane')
        self.assertEquals(foo.name.en, 'Banana')

    def test_full_clean(self):
        place = Place(name='Answer', coord_x=12.0, coord_y=42.0)
        place.full_clean()

    def test_default_kwarg(self):
        place = PlaceWithDefaultCoord()
        self.assertEqual(place.coord.x, 1.0)
        self.assertEqual(place.coord.y, 2.0)

    def test_assign_dict(self):
        place = Place(name='Answer', coord_x=12.0, coord_y=42.0)
        place.coord = {'x': 1.0, 'y': 2.0}
        self.assertEqual(place.coord.x, 1.0)
        self.assertEqual(place.coord.y, 2.0)

    def test_assign_incomplete_dict(self):
        place = Place(name='Answer', coord_x=12.0, coord_y=42.0)
        with self.assertRaises(KeyError):
            place.coord = {'x': 0.0}

    def test_bool(self):
        place = Place(name='Answer')
        self.assertFalse(place.coord)
        place.coord = {'x': 0.0, 'y': None}
        self.assertFalse(place.coord)
        place.coord = {'x': None, 'y': 0.0}
        self.assertFalse(place.coord)
        place.coord = {'x': 0.0, 'y': 0.0}
        self.assertTrue(place.coord)


class LocalizedFieldTestCase(TestCase):

    def test_general(self):
        foo = LocalizedFoo()
        self.assertEqual(len(LocalizedFoo._meta.fields), 4)
        foo.name_de = 'Mr.'
        foo.name_en = 'Herr'
        self.assertEqual(foo.name.de, 'Mr.')
        self.assertEqual(foo.name.en, 'Herr')

    def test_verbose_name(self):
        foo = LocalizedFoo()
        get_field = foo._meta.get_field
        self.assertEqual(force_text(get_field('name_de').verbose_name), 'name (de)')
        self.assertEqual(force_text(get_field('name_en').verbose_name), 'name (en)')

    def test_get_current(self):
        foo = LocalizedFoo(name_de='Bier', name_en='Beer')
```

(continues on next page)

(continued from previous page)

```
        with translation.override('de'):
            self.assertEqual(foo.name.current, 'Bier')
        with translation.override('en'):
            self.assertEqual(foo.name.current, 'Beer')

    def test_set_current(self):
        foo = LocalizedFoo()
        with translation.override('de'):
            foo.name.current = 'Bier'
        with translation.override('en'):
            foo.name.current = 'Beer'
        self.assertEqual(foo.name_de, 'Bier')
        self.assertEqual(foo.name_en, 'Beer')

    def test_set_all(self):
        foo = LocalizedFoo()
        foo.name.all = 'Felix'
        self.assertEqual(foo.name_de, 'Felix')
        self.assertEqual(foo.name_en, 'Felix')

    def test_verbose_name(self):
        foo = LocalizedFoo()
        get_field = foo._meta.get_field
        self.assertEqual(force_text(get_field('name').verbose_name), 'name')

    def test_filter(self):
        foo1 = LocalizedFoo(name_de='eins', name_en='one')
        foo2 = LocalizedFoo(name_de='zwei', name_en='two')
        try:
            foo1.save()
            foo2.save()
            with translation.override('de'):
                self.assertEqual(LocalizedFoo.objects.get(name='eins'), foo1)
                self.assertRaises(LocalizedFoo.DoesNotExist, LocalizedFoo.objects.get,
↪ name='one')
            with translation.override('en'):
                self.assertEqual(LocalizedFoo.objects.get(name='one'), foo1)
                self.assertRaises(LocalizedFoo.DoesNotExist, LocalizedFoo.objects.get,
↪ name='eins')
            with translation.override('de'):
                self.assertEqual(LocalizedFoo.objects.get(name='zwei'), foo2)
                self.assertRaises(LocalizedFoo.DoesNotExist, LocalizedFoo.objects.get,
↪ name='two')
            with translation.override('en'):
                self.assertEqual(LocalizedFoo.objects.get(name='two'), foo2)
                self.assertRaises(LocalizedFoo.DoesNotExist, LocalizedFoo.objects.get,
↪ name='zwei')
        finally:
            foo1.delete()
            foo2.delete()

    def test_order_by(self):
        foo1 = LocalizedFoo(name_de='Erdnuss', name_en='peanut')
        foo2 = LocalizedFoo(name_de='Schinken', name_en='ham')
        try:
            foo1.save()
            foo2.save()
```

(continues on next page)

(continued from previous page)

```
            with translation.override('de'):
                self.assertEqual(
                        list(LocalizedFoo.objects.all().order_by('name')),
                        [foo1, foo2])
            with translation.override('en'):
                self.assertEqual(
                        list(LocalizedFoo.objects.all().order_by('name')),
                        [foo2, foo1])
        finally:
            foo1.delete()
            foo2.delete()

    @unittest.skipIf((1, 8) <= django.VERSION < (1, 10), 'Django introduced a␣
↪infinite recursion bug for properties of deferred models that was fixed in Django 1.
↪10')
    def test_raw_sql(self):
        foo = LocalizedFoo(name_de='Antwort', name_en='answer')
        try:
            foo.save()
            foo2 = LocalizedFoo.objects.raw('SELECT * FROM composite_field_test_
↪localizedfoo')[0]
            with translation.override('de'):
                self.assertEqual(str(foo2.name), 'Antwort')
            with translation.override('en'):
                self.assertEqual(str(foo2.name), 'answer')
        finally:
            foo.delete()

    def test_bool(self):
        foo = LocalizedFoo()
        self.assertFalse(foo.name)
        foo.name_de = 'test'
        self.assertTrue(foo.name)

class ComplexFieldTestCase(TestCase):

    def test_attributes(self):
        t = ComplexTuple()
        get_field = t._meta.get_field
        self.assertEqual(get_field('x_real').blank, True)
        self.assertEqual(get_field('x_real').null, True)
        self.assertEqual(get_field('x_imag').blank, True)
        self.assertEqual(get_field('x_imag').null, True)
        self.assertEqual(get_field('y_real').blank, False)
        self.assertEqual(get_field('y_real').null, False)
        self.assertEqual(get_field('y_imag').blank, False)
        self.assertEqual(get_field('y_imag').null, False)
        self.assertEqual(get_field('z_real').blank, False)
        self.assertEqual(get_field('z_real').null, False)
        self.assertEqual(get_field('z_imag').blank, False)
        self.assertEqual(get_field('z_imag').null, False)

    def test_null(self):
        t = ComplexTuple()
        self.assertEqual(t.x, None)
        self.assertEqual(t.y, None)
        self.assertEqual(t.y, None)
```

(continues on next page)

(continued from previous page)

```
        t.x = None
        t.y = None
        t.z = None
        self.assertEqual(t.x, None)
        self.assertEqual(t.y, None)
        self.assertEqual(t.y, None)

    def test_assignment(self):
        t = ComplexTuple(x=42, y=42j, z=42+42j)
        self.assertEqual(t.x, 42)
        self.assertEqual(t.y, 42j)
        self.assertEqual(t.z, 42+42j)
        t.x = complex(21, 0)
        self.assertEqual(t.x, 21)
        t.y = complex(0, 21)
        self.assertEqual(t.y, 21j)
        t.z = complex(21, 21)
        self.assertEqual(t.z, 21+21j)

    def test_calculation(self):
        t = ComplexTuple(x=1, y=1j)
        t.z = t.x * t.y
        self.assertEqual(t.z, 1j)
        t.y *= t.y
        self.assertEqual(t.y, -1)
        t.z = t.x * t.y
        self.assertEqual(t.x, 1)
        self.assertEqual(t.y, -1)
        self.assertEqual(t.z, -1)

    def test_defaults(self):
        t = ComplexTupleWithDefaults()
        self.assertEqual(t.x, None)
        self.assertEqual(t.y, 42)
        self.assertEqual(t.z, 42j)

    def test_verbose_name(self):
        t = ComplexTuple()
        get_field = t._meta.get_field
        self.assertEqual(get_field('x_real').verbose_name, 'Re(x)')
        self.assertEqual(get_field('x_imag').verbose_name, 'Im(x)')
        self.assertEqual(get_field('y_real').verbose_name, 'Re(Y)')
        self.assertEqual(get_field('y_imag').verbose_name, 'Im(Y)')
        self.assertEqual(get_field('z_real').verbose_name, 'Re(gamma)')
        self.assertEqual(get_field('z_imag').verbose_name, 'Im(gamma)')


class InheritanceTestCase(TestCase):

    def test_abstract_inheritance(self):
        a = TranslatedModelA(name_de='Max Mustermann', name_en='John Doe')
        b = TranslatedModelB(name_en='Petra Musterfrau', name_de='Jane Doe')
        get_a_field = a._meta.get_field
        get_b_field = b._meta.get_field
        self.assertIs(get_a_field('name').model, TranslatedModelA)
        self.assertIs(get_a_field('name_de').model, TranslatedModelA)
        self.assertIs(get_a_field('name_en').model, TranslatedModelA)
```

(continues on next page)

(continued from previous page)

```
        self.assertIs(get_b_field('name').model, TranslatedModelB)
        self.assertIs(get_b_field('name_de').model, TranslatedModelB)
        self.assertIs(get_b_field('name_en').model, TranslatedModelB)

    def test_non_abstract_inheritance(self):
        c = TranslatedModelC(name_de='Max Mustermann', name_en='John Doe')
        d = TranslatedModelD(name_en='Petra Musterfrau', name_de='Jane Doe')
        get_c_field = c._meta.get_field
        get_d_field = d._meta.get_field
        self.assertIs(get_c_field('name').model, TranslatedNonAbstractBase)
        self.assertIs(get_c_field('name_de').model, TranslatedNonAbstractBase)
        self.assertIs(get_c_field('name_en').model, TranslatedNonAbstractBase)
        self.assertIs(get_d_field('name').model, TranslatedNonAbstractBase)
        self.assertIs(get_d_field('name_de').model, TranslatedNonAbstractBase)
        self.assertIs(get_d_field('name_en').model, TranslatedNonAbstractBase)


class RunChecksTestCase(TestCase):

    def test_checks(self):
        django.setup()
        from django.core import checks
        all_issues = checks.run_checks()
        errors = [str(e) for e in all_issues if e.level >= checks.ERROR]
        if errors:
            self.fail('checks failed:\n' + '\n'.join(errors))


class AdminTestCase(django.test.TestCase):

    def setUp(self):
        from django.contrib.auth.models import User
        self.factory = django.test.RequestFactory()
        self.user = self.user = User.objects.create_superuser(
                username='john.doe',
                email='john.doe@example.com',
                password='xxx12345')

    def test_login(self):
        self.assertTrue(self.client.login(username='john.doe', password='xxx12345'))

    def test_admin_index(self):
        self.client.login(username='john.doe', password='xxx12345')
        self.client.get('/admin/')

    @unittest.skipIf(django.VERSION < (1, 9), 'the admin URLs are slightly different␣
↪in django 1.9+')
    def test_translated_model_a(self):
        self.client.login(username='john.doe', password='xxx12345')
        response = self.client.get('/admin/composite_field_test/translatedmodela/')
        self.assertEquals(response.status_code, 200)
        response = self.client.get('/admin/composite_field_test/translatedmodela/add/
↪')
        self.assertEquals(response.status_code, 200)
        obj = TranslatedModelA.objects.create(name_de='Foo', name_en='Foo')
        response = self.client.get('/admin/composite_field_test/translatedmodela/')
        self.assertEquals(response.status_code, 200)
```

(continues on next page)

(continued from previous page)

```
        response = self.client.get('/admin/composite_field_test/translatedmodela/%s/
↪change/' % obj.pk)
        self.assertEquals(response.status_code, 200)
        response = self.client.get('/admin/composite_field_test/translatedmodela/%s/
↪delete/' % obj.pk)
        self.assertEquals(response.status_code, 200)

    @unittest.skipIf(django.VERSION < (1, 9), 'the admin URLs are slightly different␣
↪in django 1.9+')
    def test_crud_direction(self):
        self.client.login(username='john.doe', password='xxx12345')
        # create
        response = self.client.get('/admin/composite_field_test/direction/add/')
        self.assertEquals(response.status_code, 200)
        response = self.client.post('/admin/composite_field_test/direction/add/', {
            'source_x': '0.25',
            'source_y': '0.5',
            'distance': str(math.sqrt(2.0)),
            'target_x': '1.25',
            'target_y': '1.5',
        })
        direction = Direction.objects.get()
        self.assertEquals(direction.source_x, 0.25)
        self.assertEquals(direction.source_y, 0.5)
        self.assertAlmostEquals(direction.distance, math.sqrt(2))
        self.assertEquals(direction.target_x, 1.25)
        self.assertEquals(direction.target_y, 1.5)
        self.assertEquals(response.status_code, 302)
        # read
        response = self.client.get('/admin/composite_field_test/direction/')
        self.assertEquals(response.status_code, 200)
        response = self.client.get('/admin/composite_field_test/direction/1/change/')
        self.assertEquals(response.status_code, 200)
        # update
        response = self.client.post('/admin/composite_field_test/direction/1/change/',
↪ {
            'source_x': '0.5',
            'source_y': '0.75',
            'distance': str(math.sqrt(2.0)/2.0),
            'target_x': '1.0',
            'target_y': '1.25',
        })
        direction = Direction.objects.get()
        self.assertEquals(direction.source_x, 0.5)
        self.assertEquals(direction.source_y, 0.75)
        self.assertAlmostEquals(direction.distance, math.sqrt(2)/2.0)
        self.assertEquals(direction.target_x, 1.0)
        self.assertEquals(direction.target_y, 1.25)
        self.assertEquals(response.status_code, 302)
        # delete
        response = self.client.get('/admin/composite_field_test/direction/1/delete/')
        self.assertEquals(response.status_code, 200)
        response = self.client.post('/admin/composite_field_test/direction/1/delete/',
↪ {
            'post': 'yes',
        })
        self.assertEquals(response.status_code, 302)
```

(continues on next page)

(continued from previous page)

```
    def test_readonly(self):
        self.client.login(username='john.doe', password='xxx12345')

        place = PlaceWithDefaultCoord.objects.create()

        response = self.client.get(reverse('admin:composite_field_test_
↪placewithdefaultcoord_change', args=(place.id,)))
        self.assertEquals(response.status_code, 200)
```

CHAPTER 4

FAQ

4.1 Which Python versions are supported?

It supports Python 3.4, 3.5, 3.6, 3.7 and 3.8.

The last version to support Python 2.7 is `django-composite-field ==0.9.1`.

4.2 Which Django versions are supported?

It supports Django 1.8, 1.9, 1.10, 1.11, 2.0, 2.1, 2.2 and 3.0.

4.3 Can I get commercial support?

You can also get commercial support from the maintainer Michael P. Jung and his company Terreon GmbH.

CHAPTER 5

Changelog

5.1 Version 1.0.0, 2020-01-06

- Remove Python 2 support
- Add Django 3.0 support
- Add Documentation

5.2 Version 0.9.1, 2019-04-09

- Add Python 3.6 and 3,7 support
- Fix Django 2.2 support

5.3 Version 0.9.0, 2017-12-22

- Fix Django 2.0 private field
- Add Python 2.6 support
- Fix test_settings for Django 2.0

5.4 Version 0.8.1, 2017-10-14

- Make it simpler to provide a custom proxy
- Fix Python 2.7 compatibility for bool(LocalizedField)
- Add test for custom Proxy with __bool__()

5.5 Version 0.8.0, 2017-03-02

- Drop support for Python 3.2 and 3.3
- Add tests for modelform with include and exclude meta parameter
- Fix Python 3.x compatibility
- Improve test cases
- Add default kwarg to CompositeField
- Drop support for Django < 1.8
- Add docker configuration for running the tests
- Add bitbucket-pipeline to run tests on push

5.6 Version 0.7.6, 2016-07-01

- Add help_text, choices and max_length proxy properties
- Add support for assigning ugettext_lazy to localized fields

5.7 Version 0.7.5, 2016-06-16

- Fix Python 3 support
- Add test case for raw SQL query
- Fix ordering query sets by a localized field
- Drop support for Python 2.6
- Fix __eq__ and __lt__ method of CompositeField

5.8 Version 0.7.4, 2016-02-17

- Fix Django 1.9 support
- Add support for dicts as composite field values
- Add django-rest-framework serializer support

5.9 Version 0.7.3, 2016-01-21

- Fix Django 1.9 support
- Fix LocalizedField for Django 1.8+ when no translation is active
- Add remote_field=None to CompositeField

5.10 Version 0.7.2, 2015-11-27

- Add empty flatchoices attribute to CompositeField

5.11 Version 0.7.1, 2015-10-28

- Add primary_key=False to CompositeField

5.12 Version 0.7.0, 2015-10-26

- Fix Model.full_clean() error when using a CompositeField
- Add 'get_col' method to LocalizedField making it possible to use it in a QuerySet.

5.13 Version 0.6.0, 2015-08-21

- Fix ModelForm for models with a CompositeField
- Implement 'current(_with_default)' and 'all' property of LocalizedField

5.14 Version 0.5, 2015-07-29

- Fix composite proxy __eq__ method when comparing against non composite values
- Fix translation fallback
- Fix verbose_name as positional argument in LocalizedField

5.15 Version 0.4, 2015-07-29

- Fix Python 3.2 compatibility
- Composite field as virtual field in model

5.16 Version 0.3, 2015-07-23

- Remove deprecation warning

5.17 Version 0.2, 2015-07-23

- Add support for Django 1.4-1.8 and Python 2.x and 3.x
- Tests can be run via tox

5.18 Version 0.1, 2010-05-27

- First release

CHAPTER 6

License

```
Copyright (c) 2010-2020 Michael P. Jung
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

  * Redistributions of source code must retain the above copyright notice,
    this list of conditions and the following disclaimer.

  * Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
THE POSSIBILITY OF SUCH DAMAGE.
```

CHAPTER 7

Indices and tables

- genindex
- modindex
- search