bitfield$_{m}anager Documentation$

Release 0.3.1

Stephen Goodman

February 20, 2017

Contents

Contents:

bitfield_manager

Automatic bitfield management for Django Models.

Quickstart

Install bitfield_manager:

```
pip install django-bitfield-manager
```

Add it to your *INSTALLED_APPS*:

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

Usage

First you'll need a parent model with a status field

```
from django.db import models
from bitfield_manager.models import ParentBitfieldModel, ChildBitfieldModelMixin


class ParentExample(ParentBitfieldModel):
    status = models.BigIntegerField()

def __str__(self):  # __unicode__ on Python 2
    return "status: %i" % self.status
```

Then for all models you want django-bitfield-manager to manage add the BitfieldMeta with a list of parent models. The list of parent models takes in a tuple. The first field is the source that will be modified. The source should be a BigIntegerField or BitField (if using django-bitfield). The 2nd field is the bitflag to use (i.e. 0 will be 1 << 0, 1 will be 1 << 1, etc.)

```
class ChildExample1(ChildBitfieldModelMixin, models.Model):
    parent = models.ForeignKey('ParentExample', null=True)

    class BitfieldMeta:
        parent_models = [('parent', 'status', 0)]

class ChildExample2(ChildBitfieldModelMixin, models.Model):
    parent = models.ForeignKey('ParentExample', null=True)

    class BitfieldMeta:
        parent_models = [('parent.status', 1)]
```

Now when creating/deleting child models the parent status should update

```
# create the model
p = ParentExample.objects.create(status=0)
p2 = ParentExample.objects.create(status=0)
# add a child p.status is now 1
c1 = ChildExample1.objects.create(parent=p)

# add the other child. p.status is now 3
c2 = ChildExample2.objects.create(parent=p)

# deleting a child will refresh the status. p.status is now 2
c1.delete()

# updates or mass deletes will require manual refresh
# p.status will be 2 and p2.status will be 0
ChildExample2.objects.filter(parent=p).update(parent=p2)

# trigger a manual refresh. p.status is now correct with a status of 0
p.force_status_refresh()

# if you know the related models modified you can specify them
# p2.status is now 2
p2.force_status_refresh(related_models=[ChildExample2])

# force status refresh will work with models multiple levels deep. Specify the search_depth to searc
# more than 1 level deep
p2.force_status_refresh(search_depth=2)
```

Django Bitfield Example

```
from django.db import models
from bitfield_manager.models import ParentBitfieldModelMixin, ChildBitfieldModelMixin
from bitfield import BitField


class Person(ParentBitfieldModelMixin, models.Model):
    name = models.CharField(max_length=255)
    status = BitField(flags=(
        ('has_children', 'Has Children'),
        ('has_a_home', 'Has a Home'),
        ('has_a_car', 'Has a car')
    ))

    def __str__(self):
        return "NAME: %s STATUS: %s" % (self.name, ",".join([str(s) for s in self.status]))
```

```
class Car(ChildBitfieldModelMixin, models.Model):
    make = models.CharField(max_length=255)
    model = models.CharField(max_length=255)
    owner = models.ForeignKey('Person')

    class BitfieldMeta:
        parent_models = [('owner.status', Person.status.has_a_car)]


class Child(ChildBitfieldModelMixin, models.Model):
    name = models.CharField(max_length=255)
    parent = models.ForeignKey('Person')

    class BitfieldMeta:
        parent_models = [('parent.status', Person.status.has_children)]


class Home(ChildBitfieldModelMixin, models.Model):
    owner = models.ForeignKey('Person')

    class BitfieldMeta:
        parent_models = [('owner.status', Person.status.has_a_home)]
```

Features

- Allows for automatic bitfield management for Django Models.
- Will update the status when models are added or deleted
- Supports multi-level relationships (use dot syntax)
- Supports django-bitfield

Running Tests

Does the code actually work?

```
source <YOURVIRTUALENV>/bin/activate
(myenv) $ pip install tox
(myenv) $ tox
```

Credits

Tools used in rendering this package:

- Cookiecutter
- cookiecutter-djangopackage

CHAPTER 2

Installation

At the command line:

```
$ pip install django-bitfield-manager
```

CHAPTER 3

Usage

To use bitfield_manager in a project, add it to your *INSTALLED_APPS*:

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

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/goodmase/django-bitfield-manager/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

bitfield_manager could always use more documentation, whether as part of the official bitfield_manager docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/goodmase/django-bitfield-manager/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *django-bitfield-manager* for local development.

1. Fork the *django-bitfield-manager* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/django-bitfield-manager.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv django-bitfield-manager
$ cd django-bitfield-manager/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 bitfield_manager tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/goodmase/django-bitfield-manager/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ python -m unittest tests.test_bitfield_manager
```

CHAPTER 5

Credits

Development Lead

- Stephen Goodman <stephen.goodman@gmail.com>

Contributors

None yet. Why not be the first?

CHAPTER 6

History

0.3.0 (2017-01-31)

- Added example
- Changed the parent_models models tuple from (‘parent’, ‘child’, 0) to (‘parent.child’, 0)
- additional unit tests
- bug fixes

0.2.0 (2017-01-27)

- Added django-bitfield support
- No longer uses signals
- Added mixin for child models (ChildBitfieldModelMixin)
- Added support for one-to-one and limited support for m2m fields
- Added support for models multiple levels deep (using dot syntax)

0.1.0 (2017-01-18)

- First release on PyPI.