

---

**bitfield***manager* Documentation

**Release 0.3.0**

**Stephen Goodman**

January 31, 2017



<b>1</b>	<b>bitfield_manager</b>	<b>3</b>
1.1	Quickstart . . . . .	3
1.2	Usage . . . . .	3
1.3	Features . . . . .	4
1.4	Running Tests . . . . .	4
1.5	Credits . . . . .	5
<b>2</b>	<b>Installation</b>	<b>7</b>
<b>3</b>	<b>Usage</b>	<b>9</b>
<b>4</b>	<b>Contributing</b>	<b>11</b>
4.1	Types of Contributions . . . . .	11
4.2	Get Started! . . . . .	12
4.3	Pull Request Guidelines . . . . .	12
4.4	Tips . . . . .	13
<b>5</b>	<b>Credits</b>	<b>15</b>
5.1	Development Lead . . . . .	15
5.2	Contributors . . . . .	15
<b>6</b>	<b>History</b>	<b>17</b>
6.1	0.3.0 (2017-01-31) . . . . .	17
6.2	0.2.0 (2017-01-27) . . . . .	17
6.3	0.1.0 (2017-01-18) . . . . .	17



Contents:



---

## bitfield\_manager

---

Automatic bitfield management for Django Models.

### 1.1 Quickstart

Install `bitfield_manager`:

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

Add it to your `INSTALLED_APPS`:

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

### 1.2 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  $1 \ll 0$ , 1 will be  $1 \ll 1$ , 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 search
# more than 1 level deep
p2.force_status_refresh(search_depth=2)
```

## 1.3 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

## 1.4 Running Tests

Does the code actually work?

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



## 1.5 Credits

Tools used in rendering this package:

- Cookiecutter
- cookiecutter-djangopackage



---

## Installation

---

At the command line:

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



---

## 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:

### 4.1 Types of Contributions

#### 4.1.1 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.

#### 4.1.2 Fix Bugs

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

#### 4.1.3 Implement Features

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

#### 4.1.4 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.

#### 4.1.5 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 :)

## 4.2 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.

## 4.3 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](https://travis-ci.org/goodmase/django-bitfield-manager/pull_requests) and make sure that the tests pass for all supported Python versions.



## 4.4 Tips

To run a subset of tests:

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



---

**Credits**

---

## 5.1 Development Lead

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

## 5.2 Contributors

None yet. Why not be the first?



### 6.1 0.3.0 (2017-01-31)

- Added example
- Changed the parent\_models tuple from ('parent', 'child', 0) to ('parent.child', 0)
- additional unit tests
- bug fixes

### 6.2 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)

### 6.3 0.1.0 (2017-01-18)

- First release on PyPI.