# django-simple-history Documentation

Release 1.4.0

**Corey Bertram** 

# Contents

	1.1	umentation         Usage	
2	Code	e	
3 Changes			
	3.1	1.4.0 (2014-06-29)	
	3.2	1.3.0 (2013-05-17)	
		1.2.3 (2013-04-22)	
		1.2.1 (2013-04-22)	
		Oct 22, 2010	
		Feb 21, 2010	

django-simple-history stores Django model state on every create/update/delete.

Contents 1

2 Contents

# CHAPTER 1

**Documentation** 

### 1.1 Usage

#### 1.1.1 Install

This package is available on PyPI and Crate.io.

Install from PyPI with pip:

```
$ pip install django-simple-history
```

#### 1.1.2 Quickstart

Add simple\_history to your INSTALLED\_APPS.

To track history for a model, create an instance of simple\_history.models.HistoricalRecords on the model.

An example for tracking changes on the Poll and Choice models in the Django tutorial:

```
from django.db import models
from simple_history.models import HistoricalRecords

class Poll(models.Model):
    question = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')
    history = HistoricalRecords()

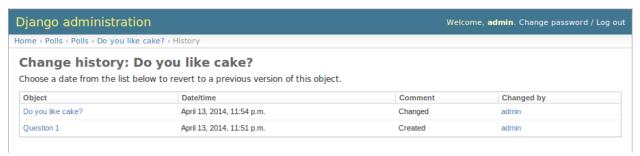
class Choice(models.Model):
    poll = models.ForeignKey(Poll)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
    history = HistoricalRecords()
```

Now all changes to Poll and Choice model instances will be tracked in the database.

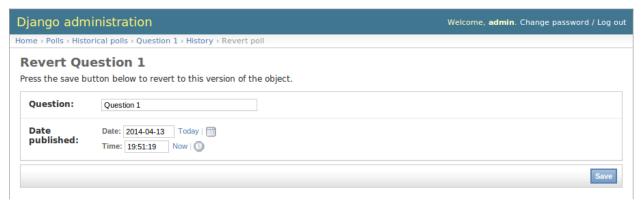
#### 1.1.3 Integration with Django Admin

To allow viewing previous model versions on the Django admin site, inherit from the simple\_history.admin. SimpleHistoryAdmin class when registering your model with the admin site.

This will replace the history object page on the admin site and allow viewing and reverting to previous model versions. Changes made in admin change forms will also accurately note the user who made the change.



Clicking on an object presents the option to revert to that version of the object.



(The object is reverted to the selected state)



Reversions like this are added to the history.



An example of admin integration for the Poll and Choice models:

```
from django.contrib import admin
from simple_history.admin import SimpleHistoryAdmin
from .models import Poll, Choice

admin.site.register(Poll, SimpleHistoryAdmin)
admin.site.register(Choice, SimpleHistoryAdmin)
```

Changing a history-tracked model from the admin interface will automatically record the user who made the change (see *Recording Which User Changed a Model*).

#### 1.1.4 Querying history

#### Querying history on a model instance

The Historical Records object on a model instance can be used in the same way as a model manager:

```
>>> from polls.models import Poll, Choice
>>> from datetime import datetime
>>> poll = Poll.objects.create(question="what's up?", pub_date=datetime.now())
>>>
>>> poll.history.all()
[<HistoricalPoll: Poll object as of 2010-10-25 18:03:29.855689>]
```

Whenever a model instance is saved a new historical record is created:

```
>>> poll.pub_date = datetime(2007, 4, 1, 0, 0)
>>> poll.save()
>>> poll.history.all()
[<HistoricalPoll: Poll object as of 2010-10-25 18:04:13.814128>, <HistoricalPoll:_

->Poll object as of 2010-10-25 18:03:29.855689>]
```

#### Querying history on a model class

Historical records for all instances of a model can be queried by using the HistoricalRecords manager on the model class. For example historical records for all Choice instances can be queried by using the manager on the Choice model class:

```
>>> choice1 = poll.choice_set.create(choice_text='Not Much', votes=0)
>>> choice2 = poll.choice_set.create(choice_text='The sky', votes=0)
>>>
>>> Choice.history
```

(continues on next page)

1.1. Usage 5

(continued from previous page)

### 1.2 Advanced Usage

#### 1.2.1 Version-controlling with South

By default, Historical models live in the same app as the model they track. Historical models are tracked by South in the same way as any other model. Whenever the original model changes, the historical model will change also.

Therefore tracking historical models with South should work automatically.

#### 1.2.2 Locating past model instance

Two extra methods are provided for locating previous models instances on historical record model managers.

#### as of

This method will return an instance of the model as it would have existed at the provided date and time.

```
>>> from datetime import datetime
>>> poll.history.as_of(datetime(2010, 10, 25, 18, 4, 0))
<HistoricalPoll: Poll object as of 2010-10-25 18:03:29.855689>
>>> poll.history.as_of(datetime(2010, 10, 25, 18, 5, 0))
<HistoricalPoll: Poll object as of 2010-10-25 18:04:13.814128>
```

#### most recent

This method will return the most recent copy of the model available in the model history.

```
>>> from datetime import datetime
>>> poll.history.most_recent()
<HistoricalPoll: Poll object as of 2010-10-25 18:04:13.814128>
```

### 1.2.3 History for Third-Party Model

To track history for a model you didn't create, use the simple\_history.register utility. You can use this to track models from third-party apps you don't have control over. Here's an example of using simple\_history.register to history-track the User model from the django.contrib.auth app:

```
from simple_history import register
from django.contrib.auth.models import User
register(User)
```

#### 1.2.4 Recording Which User Changed a Model

To denote which user changed a model, assign a \_history\_user attribute on your model.

For example if you have a changed\_by field on your model that records which user last changed the model, you could create a \_history\_user property referencing the changed\_by field:

```
from django.db import models
from simple_history.models import HistoricalRecords

class Poll(models.Model):
    question = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')
    changed_by = models.ForeignKey('auth.User')
    history = HistoricalRecords()

@property
def _history_user(self):
    return self.changed_by

@_history_user.setter
def _history_user(self, value):
    self.changed_by = value
```

Admin integration requires that you use a \_history\_user.setter attribute with your custom \_history\_user property (see *Integration with Django Admin*).

#### 1.2.5 Custom history\_date

You're able to set a custom history\_date attribute for the historical record, by defining the property \_history\_date in your model. That's helpful if you want to add versions to your model, which happened before the current model version, e.g. when batch importing historical data. The content of the property \_history\_date has to be a datetime-object, but setting the value of the property to a DateTimeField, which is already defined in the model, will work too.

```
from django.db import models
from simple_history.models import HistoricalRecords

class Poll(models.Model):
    question = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')
    changed_by = models.ForeignKey('auth.User')
    history = HistoricalRecords()
    __history_date = None

@property
def _history_date(self):
    return self.__history_date

@_history_date.setter
def _history_date(self, value):
    self.__history_date = value
```

```
from datetime import datetime
from models import Poll
```

(continues on next page)

(continued from previous page)

```
my_poll = Poll(question="what's up?")
my_poll._history_date = datetime.now()
my_poll.save()
```

#### 1.2.6 Change Base Class of Historical Record Models

To change the auto-generated HistoricalRecord models base class from models. Model, pass in the abstract class in a list to bases.

```
class RoutableModel(models.Model):
    class Meta:
        abstract = True

class Poll(models.Model):
    question = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')
    changed_by = models.ForeignKey('auth.User')
    history = HistoricalRecords(bases=[RoutableModel])
```

CHAPTER	2

Code

 ${\bf Code\ and\ issue\ tracker:\ https://github.com/treyhunner/django-simple-history}$ 

Pull requests are welcome.

10 Chapter 2. Code

# CHAPTER 3

### Changes

### 3.1 1.4.0 (2014-06-29)

- Fixed error that occurs when models have a foreign key pointing to a one to one field.
- Fix bug when model verbose\_name uses unicode (gh-76)
- Allow non-integer foreign keys
- Allow foreign keys referencing the name of the model as a string
- Added the ability to specify a custom history\_date
- Note that simple\_history should be added to INSTALLED\_APPS (gh-94 fixes gh-69)
- Properly handle primary key escaping in admin URLs (gh-96 fixes gh-81)
- Add support for new app loading (Django 1.7+)
- Allow specifying custom base classes for historical models (gh-98)

### 3.2 1.3.0 (2013-05-17)

- Fixed bug when using django-simple-history on nested models package
- Allow history table to be formatted correctly with django-admin-bootstrap
- Disallow calling simple\_history.register twice on the same model
- Added Python 3 support
- Added support for custom user model (Django 1.5+)

## 3.3 1.2.3 (2013-04-22)

• Fixed packaging bug: added admin template files to PyPI package

### 3.4 1.2.1 (2013-04-22)

- · Added tests
- Added history view/revert feature in admin interface
- Various fixes and improvements

### 3.5 Oct 22, 2010

• Merged setup.py from Klaas van Schelven - Thanks!

### 3.6 Feb 21, 2010

• Initial project creation, with changes to support ForeignKey relations.