django-reversion Documentation

Release 2.0.10

Dave Hall

Aug 18, 2017

Contents

django-reversion is an extension to the Django web framework that provides version control for model instances.

CHAPTER 1

Features

- Roll back to any point in a model instance's history.
- Recover deleted model instances.
- Simple admin integration.

CHAPTER 2

Installation

To install django-reversion:

1. Install with pip: `pip install django-reversion`.
2. Add `'reversion'` to `INSTALLED_APPS`.
3. Run `manage.py migrate`.

Important: See *Compatible Django versions* if you're not using the latest release of Django.

CHAPTER 3

Admin integration

django-reversion can be used to add rollback and recovery to your admin site.

Register your models with a subclass of *reversion.admin.VersionAdmin*.

```
from django.contrib import admin
from reversion.admin import VersionAdmin

@admin.register(YourModel)
class YourModelAdmin(VersionAdmin):

    pass
```

Hint: Whenever you register a model with django-reversion, run *createinitialrevisions*.

For more information about admin integration, see *Admin integration*.

CHAPTER 4

Low-level API

You can use the django-reversion API to build version-controlled applications. See *django-reversion API*.

CHAPTER 5

More information

Installation

Compatible Django versions

django-reversion aims to stay compatible with the latest LTS release of Django, along with more recent releases. See *django-reversion changelog*.

Older versions of Django require an older version of django-reversion to be installed.

Django version	Reversion release
1.8 - current	2.0.0
1.7	1.10.x
1.6	1.8.x

Warning: Older versions of django-reversion receive very limited support. It's advised to upgrade your Django to remain compatible with the latest release of django-reversion.

Common problems

RegistrationError: class 'myapp.MyModel' has already been registered with Reversion

This is caused by your `models.py` file being imported twice, resulting in `reversion.register()` being called twice for the same model.

This problem is almost certainly due to relative import statements in your codebase. Try converting all your relative imports into absolute imports.

django-reversion changelog

2.0.10 - 18/08/2017

- Bugfix: Handling case of *None* user in request (@pawelad).
- Documentation corrections (@danielquinn).
- Bugfix: "invalid literal for int() with base 10: 'None"' for unversioned admin inline relations.

  If, after updating, you still experience this issue, run the following in a Django shell:

```
from reversion.models import Version
Version.objects.filter(object_id="None").delete()
```

  Important: Ensure that none of your versioned models contain a string primary key where *"None"* is a valid value before running this snippet!

2.0.9 - 19/06/2017

- Bugfix: Deleted inline admin instances no longer added to revision.
- Bugfix: M2M relations correctly added to revision (@etianen, @claudep).
- Improved performance of 0003 migration (@mkurek).
- Documentation improvements (@orlra, @guettli, @meilinger).
- Django 1.11 support (@claudep).
- Added `atomic=True` parameter to `create_revision` (Ernesto Ferro).

2.0.8 - 28/11/2016

- Setting `revision.user` in `process_response` for middleware (@etianen).
- Fixing localization of model primary keys in *recover_list.html* (@w4rri0r3k).
- Documentation tweaks (@jaywink).

2.0.7 - 31/10/2016

- Database migrations now db-aware (@alukach).
- Added "revert" and "recover" context variables to admin templates (@kezabelle).
- Added `post_revision_commit` and `pre_revision_commit` signals back in (@carlosxl).
- Fixing datetime in admin change message (@arogachev).
- Fixing performance bug in postgres (@st4lk).
- Fixing admin change messages in Django 1.10+ (@claudep).
- Fixing revision middleware behavior in Django 1.10+ (@etianen).
- Documentation tweaks (@jschneier).
- Deprecation fixes (@KhasanovBI, @zsiciarz, @claudep).
- Releasing as a universal wheel (@adamchainz).

2.0.6 - 21/07/2016

- Fixed `RevisionMiddleware` always rolling back transactions in gunicorn (@stebunovd, @etianen).
- Tweaks and minor bugfixes (@SahilMak).

2.0.5 - 29/06/2016

- Fixed LookupError when running migration 0003 with stale content types (@etianen).

2.0.4 - 20/06/2016

- Fixed LookupError when running migration 0003 (@etianen).
- Fixed duplicate versions using `get_deleted()` (@etianen).
- Fixed unexpected deletion of underflowing revisions when using `--keep` switch with `deleterevisions` (@etianen).

2.0.3 - 14/06/2016

- Added support for m2m fields with a custom `through` model (@etianen).

2.0.2 - 13/06/2016

- Fixing migration 0003 in MySQL (@etianen).

2.0.1 - 13/06/2016

- Improved performance of migration 0003 (@BertrandBordage).
- De-duplicating `Version` table before applying migration 0004 (@BertrandBordage, @etianen).

2.0.0 - 11/06/2016

django-reversion was first released in May 2008, and has been in active development ever since. Over this time it's developed a certain amount of cruft from legacy and unused features, resulting in needless complexity and multiple ways of achieving the same task.

This release substantially cleans and refactors the codebase. Much of the top-level functionality remains unchanged or is very similar. The release notes are divided into subsections to make it easier to find out where you need to update your code.

This release includes a migration for the `Version` model that may take some time to complete.

General improvements

- Dramatically improved performance of version lookup for models with a non-integer primary key (@etianen, @mshannon1123).
- Documentation refactor (@etianen).
- Test refactor (@etianen).

- Minor tweaks and bugfixes (@etianen, @bmarika, @ticosax).

Admin

- Fixed issue with empty revisions being created in combination with `RevisionMiddleware` (@etianen).
- **Breaking:** Removed `reversion_format` property from `VersionAdmin` (@etianen).

  Use `VersionAdmin.reversion_register` instead.

```
class YourVersionAdmin(VersionAdmin):

    def reversion_register(self, model, **options):
        options["format"] = "yaml"
        super(YourVersionAdmin, self).reversion_register(model, **options)
```

- **Breaking:** Removed `ignore_duplicate_revisions` property from `VersionAdmin` (@etianen).

  Use `VersionAdmin.reversion_register` instead.

```
class YourVersionAdmin(VersionAdmin):

    def reversion_register(self, model, **options):
        options["ignore_duplicate_revisions"] = True
        super(YourVersionAdmin, self).reversion_register(model, **options)
```

Management commands

- **Breaking:** Refactored arguments to `createinitialrevisions` (@etianen).

  All existing functionality should still be supported, but several parameter names have been updated to match Django coding conventions.

  Check the command `--help` for details.

- **Breaking:** Refactored arguments to `deleterevisions` (@etianen).

  All existing functionality should still be supported, but several parameter names have been updated to match Django coding conventions, and some duplicate parameters have been removed. The confirmation prompt has been removed entirely, and the command now always runs in the `--force` mode from the previous version.

  Check the command `--help` for details.

Middleware

- Added support for using `RevisionMiddleware` with new-style Django 1.10 `MIDDLEWARE` (@etianen).
- Middleware wraps entire request in `transaction.atomic()` to preserve transactional integrity of revision and models (@etianen).

View helpers

- Added `reversion.views.create_revision` view decorator (@etianen).
- Added `reversion.views.RevisionMixin` class-based view mixin (@etianen).

Low-level API

- Restored many of the django-reversion API methods back to the top-level namespace (@etianen).
- Revision blocks are now automatically wrapped in `transaction.atomic()` (@etianen).
- Added `for_concrete_model` argument to `reversion.register()` (@etianen).
- Added `Version.objects.get_for_model()` lookup function (@etianen).
- Added `reversion.add_to_revision()` for manually adding model instances to an active revision (@etianen).
- Removed `Version.object_id_int` field, in favor of a unified `Version.object_id` field for all primary key types (@etianen).
- **Breaking:** `reversion.get_for_object_reference()` has been moved to `Version.objects.get_for_object_reference()` (@etianen).
- **Breaking:** `reversion.get_for_object()` has been moved to `Version.objects.get_for_object()` (@etianen).
- **Breaking:** `reversion.get_deleted()` has been moved to `Version.objects.get_deleted()` (@etianen).
- **Breaking:** `Version.object_version` has been renamed to `Version._object_version` (@etianen).
- **Breaking:** Refactored multi-db support (@etianen).

  django-reversion now supports restoring model instances to their original database automatically. Several parameter names have also be updated to match Django coding conventions.

  If you made use of the previous multi-db functionality, check the latest docs for details. Otherwise, everything should *just work*.

- **Breaking:** Removed `get_ignore_duplicates` and `set_ignore_duplicates` (@etianen).

  `ignore_duplicates` is now set in reversion.register() on a per-model basis.

- **Breaking:** Removed `get_for_date()` function (@etianen).

  Use `get_for_object().filter(revision__date_created__lte=date)` instead.

- **Breaking:** Removed `get_unique_for_object()` function (@etianen).

  Use `get_for_object().get_unique()` instead.

- **Breaking:** Removed `signal` and `eager_signals` argument from `reversion.register()` (@etianen).

  To create revisions on signals other than `post_save` and `m2m_changed`, call `reversion.add_to_revision()` in a signal handler for the appropriate signal.

```
from django.dispatch import receiver
import reversion
from your_app import your_custom_signal

@reciever(your_custom_signal)
def your_custom_signal_handler(instance, **kwargs):
    if reversion.is_active():
        reversion.add_to_revision(instance)
```

  This approach will work for both eager and non-eager signals.

- **Breaking:** Removed `adapter_cls` argument from `reversion.register()` (@etianen).
- **Breaking:** Removed `reversion.save_revision()` (@etianen).

  Use reversion.add_to_revision() instead.

```
import reversion

with reversion.create_revision():
    reversion.add_to_revision(your_obj)
```

Signals

- **Breaking:** Removed `pre_revision_commit` signal (@etianen).

  Use the Django standard `pre_save` signal for `Revision` instead.
- **Breaking:** Removed `post_revision_commit` signal (@etianen).

  Use the Django standard `post_save` signal for `Revision` instead.

Helpers

- **Breaking:** Removed `patch_admin` function (@etianen).

  Use `VersionAdmin` as a mixin to 3rd party ModelAdmins instead.

```
@admin.register(SomeModel)
class YourModelAdmin(VersionAdmin, SomeModelAdmin):

    pass
```

- **Breaking:** Removed `generate_diffs` function (@etianen).

  django-reversion no supports an official diff helper. There are much better ways of achieving this now, such as django-reversion-compare.

  The old implementation is available for reference from the previous release.
- **Breaking:** Removed `generate_patch` function (@etianen).

  django-reversion no supports an official diff helper. There are much better ways of achieving this now, such as django-reversion-compare.

  The old implementation is available for reference from the previous release.
- **Breaking:** Removed `generate_patch_html` function (@etianen).

  django-reversion no supports an official diff helper. There are much better ways of achieving this now, such as django-reversion-compare.

  The old implementation is available for reference from the previous release.

Models

- **Breaking:** Ordering of `-pk` added to models `Revision` and `Version`. Previous was the default `pk`.

1.10.2 - 18/04/2016

- Fixing deprecation warnings (@claudep).
- Minor tweaks and bug fixes (@fladi, @claudep, @etianen).

1.10.1 - 27/01/2016

- Fixing some deprecation warnings (@ticosax).
- Minor tweaks (@claudep, @etianen).

1.10 - 02/12/2015

- **Breaking:** Updated the location of `VersionAdmin`.

  Prior to this change, you could access the `VersionAdmin` class using the following import:

  ```
  # Old-style import for accessing the admin class.
  import reversion

  # Access admin class from the reversion namespace.
  class YourModelAdmin(reversion.VersionAdmin):

      pass
  ```

  In order to support Django 1.9, the admin class has been moved to the following import:

  ```
  # New-style import for accesssing admin class.
  from reversion.admin import VersionAdmin

  # Use the admin class directly.
  class YourModelAdmin(VersionAdmin):

      pass
  ```

- **Breaking: Updated the location of low-level API methods.** Prior to this change, you could access the low-level API using the following import:

  ```
  # Old-style import for accessing the low-level API.
  import reversion

  # Use low-level API methods from the reversion namespace.
  @reversion.register
  class YourModel(models.Model):

      pass
  ```

  In order to support Django 1.9, the low-level API methods have been moved to the following import:

  ```
  # New-style import for accesssing the low-level API.
  from reversion import revisions as reversion

  # Use low-level API methods from the revisions namespace.
  @reversion.register
  class YourModel(models.Model):
  ```

```
    pass
```

- **Breaking: Updated the location of http://django-reversion.readthedocs.org/en/latest/signals.html.** Prior to this change, you could access the reversion signals using the following import:

```
# Old-style import for accessing the reversion signals
import reversion

# Use signals from the reversion namespace.
reversion.post_revision_commit.connect(...)
```

  In order to support Django 1.9, the reversion signals have been moved to the following import:

```
# New-style import for accesssing the reversion signals.
from reversion.signals import pre_revision_commit, post_revision_commit

# Use reversion signals directly.
post_revision_commit.connect(...)
```

- Django 1.9 compatibility (@etianen).
- Added spanish (argentina) translation (@gonzalobustos).
- Minor bugfixes and tweaks (@Blitzstok, @IanLee1521, @lutoma, @siamalekpour, @etianen).

1.9.3 - 07/08/2015

- Fixing regression with admin redirects following save action (@etianen).

1.9.2 - 07/08/2015

- Fixing regression with "delete", "save as new" and "save and continue" button being shown in recover and revision admin views (@etianen).
- Fixing regression where VersionAdmin.ignore_duplicate_revisions was ignored (@etianen).

1.9.1 - 04/08/2015

- Fixing packaging error that rendered the 1.9.0 release unusable. No way to cover up the mistake, so here's a brand new bugfix release! (@etianen).

1.9.0 - 04/08/2015

- Using database transactions do render consistent views of past revisions in database admin, fixing a lot of lingering minor issues (@etianen).
- Correct handling of readonly fields in admin (@etianen).
- Updates to Czech translation (@cuchac).
- Arabic translation (@RamezIssac).
- Fixing deleterevisions to work with Python2 (@jmurty).
- Fixing edge-cases where an object does not have a PK (@johnfraney).

- Tweaks, code cleanups and documentation fixes (@claudep, @johnfraney, @podloucky-init, Drew Hubl, @Jan-Malte, @jmurty, @etianen).

1.8.7 - 21/05/2015

- Fixing deleterevisions command on Python 3 (@davidfsmith).
- Fixing Django 1.6 compatibility (@etianen).
- Removing some Django 1.9 deprecation warnings (@BATCOH, @niknokseyer).
- Minor tweaks (@nikolas, @etianen).

1.8.6 - 13/04/2015

- Support for MySQL utf8mb4 (@alexhayes).
- Fixing some Django deprecation warnings (Drew Hubl, @khakulov, @adonm).
- Versions passed through by reversion.post_revision_commit now contain a primary key (@joelarson).

1.8.5 - 31/10/2014

- Added support for proxy models (@AgDude, @bourivouh).
- Allowing registration of models with django-reversion using custom signals (@ErwinJunge).
- Fixing some Django deprecation warnings (@skipp, @narrowfail).

1.8.4 - 07/09/2014

- Fixing including legacy south migrations in PyPi package (@GeyseR).

1.8.3 - 06/09/2014

- Provisional Django 1.7 support (@etianen).
- Multi-db and multi-manager support to management commands (@marekmalek).
- Added index on reversion.date_created (@rkojedzinszky).
- Minor bugfixes and documentation improvements (@coagulant).

1.8.2 - 01/08/2014

- reversion.register() can now be used as a class decorator (@aquavitae).
- Danish translation (@Vandborg).
- Improvements to Travis CI integration (@thedrow).
- Simplified Chinese translation (@QuantumGhost).
- Minor bugfixes and documentation improvements (@marekmalek, @dhoffman34, @mauricioabreu, @mark0978).

1.8.1 - 29/05/2014

- Slovak translation (@jbub).
- Deleting a user no longer deletes the associated revisions (@daaray).
- Improving handling of inline models in admin integration (@blueyed).
- Improving error messages for proxy model registration (@blueyed).
- Improvements to using migrations with custom user model (@aivins).
- Removing sys.exit() in deleterevisions management command, allowing it to be used internally by Django projects (@tongwang).
- Fixing some backwards-compatible admin deprecation warnings (Thomas Schreiber).
- Fixing tests if RevisionMiddleware is used as a decorator in the parent project (@jmoldow).
- Derived models, such as those generated by deferred querysets, now work.
- Removed deprecated low-level API methods.

1.8.0 - 01/11/2013

- Django 1.6 compatibility (@niwibe & @meshy).
- Removing type flag from Version model.
- Using bulk_create to speed up revision creation.
- Including docs in source distribution (@pquentin & @fladi).
- Spanish translation (@alexander-ae).
- Fixing edge-case bugs in revision middleware (@pricem & @oppianmatt).

1.7.1 - 26/06/2013

- Bugfixes when using a custom User model.
- Minor bugfixes.

1.7 - 27/02/2013

- Django 1.5 compatibility.
- Experimantal Python 3.3 compatibility!

1.6.6 - 12/02/2013

- Removing version checking code. It's more trouble than it's worth.
- Dutch translation improvements.

1.6.5 - 12/12/2012

- Support for Django 1.4.3.

1.6.4 - 28/10/2012

- Support for Django 1.4.2.

1.6.3 - 05/09/2012

- Fixing issue with reverting models with unique constraints in the admin.
- Enforcing permissions in admin views.

1.6.2 - 31/07/2012

- Batch saving option in createinitialrevisions.
- Suppressing warning for Django 1.4.1.

1.6.1 - 20/06/2012

- Swedish translation.
- Fixing formating for PyPi readme and license.
- Minor features and bugfixes.

1.6 - 27/03/2012

- Django 1.4 compatibility.

1.5.2 - 27/03/2012

- Multi-db support.
- Brazillian Portuguese translation.
- New manage_manually revision mode.

1.5.1 - 20/10/2011

- Polish translation.
- Minor bug fixes.

1.5 - 04/09/2011

- Added in simplified low level API methods, and deprecated old low level API methods.
- Added in support for multiple revision managers running in the same project.
- Added in significant speedups for models with integer primary keys.
- Added in cleanup improvements to patch generation helpers.
- Minor bug fixes.

1.4 - 27/04/2011

- Added in a version flag for add / change / delete annotations.
- Added experimental deleterevisions management command.
- Added a –comment option to createinitialrevisions management command.
- Django 1.3 compatibility.

1.3.3 - 05/03/2011

- Improved resilience of revert() to database integrity errors.
- Added in Czech translation.
- Added ability to only save revisions if there is no change.
- Fixed long-running bug with file fields in inline related admin models.
- Easier debugging for createinitialrevisions command.
- Improved compatibility with Oracle database backend.
- Fixed error in MySQL tests.
- Greatly improved performance of get_deleted() Version manager method.
- Fixed an edge-case UnicodeError.

1.3.2 - 22/10/2010

- Added Polish translation.
- Added French translation.
- Improved resilience of unit tests.
- Improved scaleability of Version.object.get_deleted() method.
- Improved scaleability of createinitialrevisions command.
- Removed post_syncdb hook.
- Added new createinitialrevisions management command.
- Fixed DoesNotExistError with OneToOneFields and follow.

1.3.1 - 31/05/2010

This release is compatible with Django 1.2.1.

- Django 1.2.1 admin compatibility.

1.2.1 - 03/03/2010

This release is compatible with Django 1.1.1.

- The django syncdb command will now automatically populate any version-controlled models with an initial revision. This ensures existing projects that integrate Reversion won't get caught out.
- Reversion now works with SQLite for tables over 999 rows.

- Added Hebrew translation.

1.2 - 12/10/2009

This release is compatible with Django 1.1.

- Django 1.1 admin compatibility.

1.1.2 - 23/07/2009

This release is compatible with Django 1.0.4.

- Doc tests.
- German translation update.
- Better compatibility with the Django trunk.
- The ability to specify a serialization format used by the ReversionAdmin class when models are auto-registered.
- Reduction in the number of database queries performed by the Reversion
- admin interface.

1.1.1 - 25/03/2010

This release is compatible with Django 1.0.2.

- German and Italian translations.
- Helper functions for generating diffs.
- Improved handling of one-to-many relationships in the admin.

Usage

Admin integration

django-reversion can be used to add rollback and recovery to your admin site.

> **Warning:** The admin integration requires that your database engine supports transactions. This is the case for PostgreSQL, SQLite and MySQL InnoDB. If you are using MySQL MyISAM, upgrade your database tables to InnoDB!

Overview

Registering models

Register your models with a subclass of *reversion.admin.VersionAdmin*.

```
from django.contrib import admin
from reversion.admin import VersionAdmin

@admin.register(YourModel)
class YourModelAdmin(VersionAdmin):

    pass
```

Hint: Whenever you register a model with django-reversion, run *createinitialrevisions*.

Note: If you've registered your models using *reversion.register()*, the admin class will use the configuration you specify there. Otherwise, the admin class will auto-register your model, following all inline model relations and parent superclasses.

Integration with 3rd party apps

You can use *reversion.admin.VersionAdmin* as a mixin with a 3rd party admin class.

```
@admin.register(SomeModel)
class YourModelAdmin(VersionAdmin, SomeModelAdmin):

    pass
```

If the 3rd party model is already registered with the Django admin, you may have to unregister it first.

```
admin.site.unregister(SomeModel)

@admin.register(SomeModel)
class YourModelAdmin(VersionAdmin, SomeModelAdmin):

    pass
```

reversion.admin.VersionAdmin

A subclass of `django.contrib.ModelAdmin` providing rollback and recovery.

`revision_form_template = None`

A custom template to render the revision form.

Alternatively, create specially named templates to override the default templates on a per-model or per-app basis.

- `'reversion/app_label/model_name/revision_form.html'`
- `'reversion/app_label/revision_form.html'`
- `'reversion/revision_form.html'`

`recover_list_template = None`

A custom template to render the recover list.

Alternatively, create specially named templates to override the default templates on a per-model or per-app basis.

- `'reversion/app_label/model_name/recover_list.html'`
- `'reversion/app_label/recover_list.html'`
- `'reversion/recover_list.html'`

`recover_form_template = None`

A custom template to render the recover form.

- `'reversion/app_label/model_name/recover_form.html'`
- `'reversion/app_label/recover_form.html'`
- `'reversion/recover_form.html'`

`history_latest_first = False`

If `True`, revisions will be displayed with the most recent revision first.

`reversion_register(model, **options)`

Callback used by the auto-registration machinery to register the model with django-reversion. Override this to customize how models are registered.

```
def reversion_register(self, model, **options):
    options["exclude"] = ("some_field",)
    super(YourModelAdmin, self).reversion_register(model, **options)
```

`model` The model that will be registered with django-reversion.

`options` Registeration options, see *reversion.register()*.

Management commands

django-reversion includes a number of `django-admin.py` management commands.

createinitialrevisions

Creates an initial revision for all registered models in your project. It should be run after installing django-reversion, or registering a new model with django-reversion.

```
./manage.py createinitialrevisions
./manage.py createinitialrevisions your_app.YourModel --comment="Initial revision."
```

Run `./manage.py createinitialrevisions --help` for more information.

Warning: For large databases, this command can take a long time to run.

deleterevisions

Deletes old revisions. It can be run regularly to keep revision history manageable.

```
./manage.py deleterevisions
# keep any changes from last 30 days
./manage.py deleterevisions your_app.YourModel --days=30
# keep 30 most recent changes for each item.
./manage.py deleterevisions your_app.YourModel --keep=30
# Keep anything from last 30 days and at least 3 from older changes.
./manage.py deleterevisions your_app.YourModel --keep=3 --days=30
```

Run `./manage.py deleterevisions --help` for more information.

Warning: With no arguments, this command will delete your entire revision history! Read the command help for ways to limit which revisions should be deleted.

django-reversion API

Use the django-reversion API to build version-controlled apps. See also *Views* and *Middleware*.

Overview

Registering models

Models must be registered with django-reversion before they can be used with the API.

```
from django.db import models
import reversion

@reversion.register()
class YourModel(models.Model):

    pass
```

Hint: If you're using the *Admin integration*, model registration is automatic. If you're using django-reversion in a management command, make sure you call `django.contrib.admin.autodiscover()` to load the admin modules before using the django-reversion API.

Hint: Whenever you register a model with django-reversion, run *createinitialrevisions*.

Creating revisions

A *revision* represents one or more changes made to your model instances, grouped together as a single unit. You create a revision by creating a *revision block*. When you call `save()` on a registered model inside a revision block, it will be added to that revision.

```
# Declare a revision block.
with reversion.create_revision():

    # Save a new model instance.
```

```
    obj = YourModel()
    obj.name = "obj v1"
    obj.save()

    # Store some meta-information.
    reversion.set_user(request.user)
    reversion.set_comment("Created revision 1")

# Declare a new revision block.
with reversion.create_revision():

    # Update the model instance.
    obj.name = "obj v2"
    obj.save()

    # Store some meta-information.
    reversion.set_user(request.user)
    reversion.set_comment("Created revision 2")
```

Important: Bulk actions, such as `Queryset.update()`, do not send signals, so won't be noticed by django-reversion.

Loading revisions

Each model instance saved in a revision block is serialized as a *reversion.models.Version*. All versions in a revision block are associated with a single *reversion.models.Revision*.

You can load a *reversion.models.VersionQuerySet* of versions from the database. Versions are loaded with the most recent version first.

```
from reversion.models import Version

# Load a queryset of versions for a specific model instance.
versions = Version.objects.get_for_object(instance)
assert len(versions) == 2

# Check the serialized data for the first version.
assert versions[1].field_dict["name"] = "obj v1"

# Check the serialized data for the second version.
assert versions[0].field_dict["name"] = "obj v2"
```

Revision metadata

reversion.models.Revision stores meta-information about the revision.

```
# Check the revision metadata for the first revision.
assert versions[1].revision.comment = "Created revision 1"
assert versions[1].revision.user = request.user
assert isinstance(versions[1].revision.date_created, datetime.datetime)

# Check the revision metadata for the second revision.
assert versions[0].revision.comment = "Created revision 2"
```

```
assert versions[0].revision.user = request.user
assert isinstance(versions[0].revision.date_created, datetime.datetime)
```

Reverting revisions

Revert a *reversion.models.Revision* to restore the serialized model instances.

```
# Revert the first revision.
versions[1].revision.revert()

# Check the model instance has been reverted.
obj.refresh_from_db()
assert obj.name == "version 1"

# Revert the second revision.
versions[0].revision.revert()

# Check the model instance has been reverted.
obj.refresh_from_db()
assert obj.name == "version 2"
```

Restoring deleted model instances

Reverting a *reversion.models.Revision* will restore any serialized model instances that have been deleted.

```
# Delete the model instance, but store the pk.
pk = obj.pk
obj.delete()

# Revert the second revision.
versions[0].revision.revert()

# Check the model has been restored to the database.
obj = YourModel.objects.get(pk=obj.pk)
assert obj.name == "version 2"
```

Registration API

`reversion.register(model, **options)`

Registers a model with django-reversion.

Throws *reversion.RegistrationError* if the model has already been registered.

`model` The Django model to register.

`fields=None` An iterable of field names to include in the serialized data. If `None`, all fields will be included.

`exclude=()` An iterable of field names to exclude from the serialized data.

`follow=()` An iterable of model relationships to follow when saving a version of this model. `ForeignKey`, `ManyToManyField` and reversion `ForeignKey` relationships are supported. Any property that returns a `Model` or `QuerySet` is also supported.

format="json" The name of a Django serialization format to use when saving the model instance.

for_concrete_model=True If `True` proxy models will be saved under the same content type as their concrete model. If `False`, proxy models will be saved under their own content type, effectively giving proxy models their own distinct history.

ignore_duplicates=False If `True`, then an additional check is performed to avoid saving duplicate versions for this model.

Checking for duplicate revisions adds significant overhead to the process of creating a revision. Don't enable it unless you really need it!

Hint: By default, django-reversion will not register any parent classes of a model that uses multi-table inheritance. If you wish to also add parent models to your revision, you must explicitly add their `parent_ptr` fields to the `follow` parameter when you register the model.

Hint: Whenever you register a model with django-reversion, run *createinitialrevisions*.

`reversion.is_registered(model)`

Returns whether the given model has been registered with django-reversion.

model The Django model to check.

`reversion.unregister(model)`

Unregisters the given model from django-reversion.

Throws *reversion.RegistrationError* if the model has not been registered with django-reversion.

model The Django model to unregister.

`reversion.get_registered_models()`

Returns an iterable of all registered models.

Revision API

`reversion.create_revision(manage_manually=False, using=None, atomic=True)`

Marks a block of code as a *revision block*. Can also be used as a decorator.

manage_manually If `True`, versions will not be saved when a model's `save()` method is called. This allows version control to be switched off for a given revision block.

using The database to save the revision data. The revision block will be wrapped in a transaction using this database. If `None`, the default database for *reversion.models.Revision* will be used.

atomic If `True`, the revision block will be wrapped in a `transaction.atomic()`.

`reversion.is_active()`

Returns whether there is currently an active revision block.

`reversion.is_manage_manually()`

Returns whether the current revision block is in `manage_manually` mode.

`reversion.set_user(user)`

Sets the user for the current revision.

Throws *reversion.RevisionManagementError* if there is no active revision block.

user A `User` model instance (or whatever your `settings.AUTH_USER_MODEL` is).

`reversion.get_user()`

Returns the user for the current revision.

Throws *reversion.RevisionManagementError* if there is no active revision block.

`reversion.set_comment(comment)`

Sets the comment for the current revision.

Throws *reversion.RevisionManagementError* if there is no active revision block.

comment The text comment for the revision.

`reversion.get_comment()`

Returns the comment for the current revision.

Throws *reversion.RevisionManagementError* if there is no active revision block.

`reversion.set_date_created(date_created)`

Sets the creation date for the current revision.

Throws *reversion.RevisionManagementError* if there is no active revision block.

date_created The creation date for the revision.

`reversion.get_date_created()`

Returns the creation date for the current revision.

Throws *reversion.RevisionManagementError* if there is no active revision block.

`reversion.add_meta(model, **values)`

Adds custom metadata to a revision.

Throws *reversion.RevisionManagementError* if there is no active revision block.

model A Django model to store the custom metadata. The model must have a `ForeignKey` or `OneToOneField` to *reversion.models.Revision*.

****values** Values to be stored on `model` when it is saved.

`reversion.add_to_revision(obj, model_db=None)`

Adds a model instance to a revision.

Throws *reversion.RevisionManagementError* if there is no active revision block.

obj A model instance to add to the revision.

model_db The database where the model is saved. Defaults to the default database for the model.

reversion.models.VersionQuerySet

A `QuerySet` of *reversion.models.Version*. The results are ordered with the most recent *reversion.models.Version* first.

`Version.objects.get_for_model(model, model_db=None)`

Returns a *reversion.models.VersionQuerySet* for the given model.

Throws *reversion.RegistrationError* if the model has not been registered with django-reversion.

model A registered model.

model_db The database where the model is saved. Defaults to the default database for the model.

`Version.objects.get_for_object(obj, model_db=None)`

Returns a *reversion.models.VersionQuerySet* for the given model instance.

Throws *reversion.RegistrationError* if the model has not been registered with django-reversion.

obj An instance of a registered model.

model_db The database where the model is saved. Defaults to the default database for the model.

`Version.objects.get_for_object_reference(model, pk, model_db=None)`

Returns a *reversion.models.VersionQuerySet* for the given model and primary key.

Throws *reversion.RegistrationError* if the model has not been registered with django-reversion.

model A registered model.

pk The database primary key of a model instance.

model_db The database where the model is saved. Defaults to the default database for the model.

`Version.objects.get_deleted(model, model_db=None)`

Returns a *reversion.models.VersionQuerySet* for the given model containing versions where the serialized model no longer exists in the database.

Throws *reversion.RegistrationError* if the model has not been registered with django-reversion.

model A registered model.

db The database to load the versions from.

model_db The database where the model is saved. Defaults to the default database for the model.

`Version.objects.get_unique()`

Returns an iterable of *reversion.models.Version*, where each version is unique for a given database, model instance, and set of serialized fields.

reversion.models.Version

Represents a single model instance serialized in a revision.

`Version.id`

The database primary key of the *reversion.models.Version*.

`Version.revision`

A `ForeignKey` to a *reversion.models.Revision* instance.

`Version.content_type`

The `ContentType` of the serialized model instance.

`Version.object_id`

The string representation of the serialized model instance's primary key.

`Version.db`

The Django database alias where the serialized model was saved.

`Version.format`

The name of the Django serialization format used to serialize the model instance.

`Version.serialized_data`

The raw serialized data of the model instance.

`Version.object_repr`

The stored snapshot of the model instance's `__str__` method when the instance was serialized.

`Version.field_dict`

A dictionary of stored model fields. This includes fields from any parent models in the same revision.

Throws *reversion.RevertError* if the model could not be deserialized or reverted, e.g. the serialized data is not compatible with the current database schema.

`Version.revert()`

Restores the serialized model instance to the database. To restore the entire revision, use *Revision.revert()*.

Throws *reversion.RevertError* if the model could not be deserialized or reverted, e.g. the serialized data is not compatible with the current database schema.

reversion.models.Revision

Contains metadata about a revision, and groups together all *reversion.models.Version* instances created in that revision.

`Revision.id`

The database primary key of the *reversion.models.Revision*.

`Revision.date_created`

A `datetime` when the revision was created.

`Revision.user`

The `User` that created the revision, or None.

`Revision.comment`

A text comment on the revision.

`Revision.revert(delete=False)`

Restores all contained serialized model instances to the database.

Throws *reversion.RevertError* if the model could not be deserialized or reverted, e.g. the serialized data is not compatible with the current database schema.

`delete` If `True`, any model instances which have been created and are reachable by the `follow` clause of any model instances in this revision will be deleted. This effectively restores a group of related models to the state they were in when the revision was created.

Views

Shortcuts when using django-reversion in views.

Decorators

`reversion.views.create_revision(manage_manually=False, using=None, atomic=True)`

Decorates a view to wrap every request that isn't `GET`, `HEAD` or `OPTIONS` in a revision block.

The request user will also be added to the revision metadata. You can set the revision comment by calling *reversion.set_comment()* within your view.

`manage_manually` If `True`, versions will not be saved when a model's `save()` method is called. This allows version control to be switched off for a given revision block.

`using` The database to save the revision data. The revision block will be wrapped in a transaction using this database. If `None`, the default database for *reversion.models.Revision* will be used.

`atomic` If `True`, the revision block will be wrapped in a `transaction.atomic()`.

reversion.views.RevisionMixin

Mixin a class-based view to wrap every request that isn't `GET`, `HEAD` or `OPTIONS` in a revision block.

The request user will also be added to the revision metadata. You can set the revision comment by calling *reversion.set_comment()* within your view.

```
from django.contrib.auth.views import FormView
from reversion.views import RevisionMixin

class RevisionFormView(RevisionMixin, FormView):

    pass
```

`RevisionMixin.revision_manage_manually = False`

If `True`, versions will not be saved when a model's `save()` method is called. This allows version control to be switched off for a given revision block.

`RevisionMixin.revision_using = None`

The database to save the revision data. The revision block will be wrapped in a transaction using this database. If `None`, the default database for *reversion.models.Revision* will be used.

Middleware

Shortcuts when using django-reversion in views.

reversion.middleware.RevisionMiddleware

Wrap the every request that isn't `GET`, `HEAD` or `OPTIONS` in a revision block.

The request user will also be added to the revision metadata.

To enable `RevisionMiddleware`, add `'reversion.middleware.RevisionMiddleware'` to your `MIDDLEWARE_CLASSES` setting. For Django >= 1.10, add it to your `MIDDLEWARE` setting.

Warning: This will wrap every request that isn't `GET`, `HEAD` or `OPTIONS` in a database transaction. For best performance, consider marking individual views instead.

`RevisionMiddleware.manage_manually = False`

If `True`, versions will not be saved when a model's `save()` method is called. This allows version control to be switched off for a given revision block.

`RevisionMiddleware.using = None`

The database to save the revision data. The revision block will be wrapped in a transaction using this database. If `None`, the default database for *reversion.models.Revision* will be used.

`RevisionMiddleware.atomic = True`

If `True`, the revision block will be wrapped in a `transaction.atomic()`.

Errors

django-reversion defines several custom errors.

reversion.RegistrationError

Something went wrong with the *Registration API*.

reversion.RevisionManagementError

Something went wrong using the *Revision API*.

reversion.RevertError

Something went wrong reverting a revision.

Signals

django-reversion provides two custom signals.

reversion.signals.pre_revision_commit

Sent just before a revision is saved to the database.

sender The `reversion.create_revision` object.

revision The *reversion.models.Revision* model.

versions The *reversion.models.Version* models in the revision.

reversion.signals.post_revision_commit

Sent just after a revision and its related versions are saved to the database.

sender The `reversion.create_revision` object.

revision The *reversion.models.Revision* model.

versions The *reversion.models.Version* models in the revision.