Aldryn Reversion Documentation

Release 1.1.0

Divio AG

January 10, 2018

Contents

Aldryn Reversion is an Aldryn-compatible application for for django CMS, containing a collection of shared helpers and mixins to provide support for django-reversion on models with translatable (using django-parler) fields as well as django CMS placeholder fields.

Aldryn Reversion allows you to register your application models with django-reversion along with additional tools. It extends the functionality provided by django-reversion to include Placeholders, CMS Plugins and translated fields into recoverable history.

Aldryn Reversion is open-source software.

CHAPTER 1

Introduction

Installation

Most likely you won't need to install this addon - it will be installed as a dependency for some other addon.

Installing packages

Then run either:

```
pip install aldryn-reversion
```

or to install from the latest source tree:

```
pip install -e git+https://github.com/aldryn/aldryn-reversion.git#egg=aldryn-reversion
```

Note: This should also install `django-reversion` (which is required) into your env.

You'll get the most benefit from aldryn-reversion if you're using it with django CMS and django-parler. We'll assume you have a django CMS (version 3.x) project up and running.

If you need to set up a new django CMS project, follow the instructions in the django CMS tutorial.

In this case `django-parler` should be installed as a dependency for django CMS.

`settings.py`

Edit your `INSTALLED_APPS` to include the required applications:

```
INSTALLED_APPS = [
    ...
    'reversion',
    'aldryn_reversion',
    ...
]
```

Note: `reversion` is required, but please check that it is included into `INSTALLED_APPS` only once, since it might be already there if you are using django CMS.

If you are planning to use the optional packages add them to `INSTALLED_APPS` too.

Todo

say something about what these optional packages are

Prepare the database and run

Now run `python manage.py migrate` to prepare the database for the new application, then `python manage.py runserver`.

For Aldryn users

If you are using *any* addon that relies on `aldryn-reversion` it means that you already have `aldryn-reversion` installed and configured.

If you are developing an Aldryn addon and want to use `aldryn-reversion` features in it, please refer to Apply Aldryn Reversion support

Apply Aldryn Reversion support

Using Aldryn Reversion in your project is fairly strightforward.

There are two key steps required:

- applying a registration decorator to your model
- adding a mixin to the model's admin class definition

Model registration

For the model, add `@version_controlled_content` as a decorator:

```
from aldryn-reversion.core import version_controlled_content

@version_controlled_content
class MyModel(models.Model):
    ...
```

If your model has foreign key relations, use the `follow` property from `django-reversions` when applying the decorator:

```
@version_controlled_content(follow=['my_fk_field'])
class MyModel(models.Model):
    my_fk_field = models.ForeignKey(OtherModel)
    ...
```

This also adds the related objects as version-controlled content.

Note: These related models should also be registered with django-reversion in their own right.

Options

If you need more control over exactly what is versioned, then use `@reversion.register()` rather than the `@version-controlled-content` decorator.

follow_placeholders

By default, a model's Placeholder fields are versioned (in case of using `@version-controlled-content` decorator). Setting `follow_placeholders` to `False` disables this behaviour. In this case *changes to plugins inside this placeholder* fields are not revisioned but if we change the placeholder object this field points to, that change will be picked up by reversion.

Pass this option when registering the model with reversion:

```
@reversion.register(
    adapter_cls=ContentEnabledVersionAdapter,
    follow_placeholders=False,
    revision_manager=reversion.default_revision_manager,
)
class MyModel(models.Model):
    ...
    placeholder = PlaceholderField()
```

follow

If your model has foreign key relations, whose destination objects you also want versioned, use the `follow` property from `django-reversions` when applying the decorator:

```
@version_controlled_content(follow=['my_fk_relation', 'other_fk_relation'])
class MyModel(models.Model):
    ...
    my_fk_relation = models.ForeignKey(OtherModel)
    other_fk_relation = models.ForeignKey(OtherModel)
```

Other options

You should be able to also use other `django-reversion` options that are available for `revision.register` as described in Advanced model registration

Admin registration

For the admin, replace `PlaceholderAdminMixin` with `VersionedPlaceholderAdminMixin` in the `ModelAdmin` class for any models that include Placeholders that need to be versioned:

```
from aldryn-reversion.admin import VersionedPlacholderAdminMixin

class MyModelAdmin(VersionedPlacholderAdminMixin, admin.ModelAdmin):
    ...
```

Revisions are accessible from the model's admin change form.

Important: In restoring a revision you will **also** restore all objects that belong to that revision to the state in which they were saved with that revision. This behaviour may not be expected by end-users.

Deleted objects

If an object has been deleted, its admin change form will obviously no longer be available.

However, the model's admin change list view offers a **Recover view**, that allows you to restore a deleted object along with the translations that belong to it.

If foreign key relations have been registered with the `follow` property and they are required for this object, they too will be restored automatically, to the state captured in the relevant revision.

CHAPTER 2

Using Aldryn Reversion

After you have set up and configured your models so that they are registered for revision support you can start using the `aldryn-reversion` end-user features.

Restoring object versions

Django already keeps track of changes to objects and makes their history available in the object's qadmin edit form. With `aldryn-reversion` that history has more features.

As soon as an object is created, an *Initial version* is created and treated as a restore point.

For each record in an object's history you'll now find a link to the *restore revision* form.

To restore a previous object version, select the desired date, check what will be changed, and if you are satisfied with the proposed changes, confirm to restore the revision.

Note: Note that the revision will be restored *as is* - all objects related to the object you are restoring will also be restored to the same revision. That means that any related object that is also stored in this revision would be restored to the point at which that object was stored.

This will be applicable to entire relation tree (if `follow` was configured).

Once the the object is restored its content will be set to the revision you've chosen and you will be redirected to the object's edit form.

You might also want to modify some of the related objects, since they too will be affected by restoring to a previous revision.

Note: If *follow* was not configured properly and as a result the related objects were not stored as part of the selected revision, you could end up with an `IntegrityError`. In such a case you will first need to restore the related object.

Recovering deleted objects

Another powerful feature of `aldryn-reversion` is the ability to recover deleted objects.

This function is available on a model's change list page - a *Recover deleted* option is available near the top of the page.

If there are any deleted objects which do not currently exist in the database but for which there is a version stored by reversions, they will be listed on the recovery page.

To recover a deleted object, simply choose an object, check the instructions and if there are no conflicts for the related fields select the **Yes, I'm sure** button.

If there are any conflicts that can be resolved by the user (the model was registered with the `aldryn-reversions` decorator and with admin mixin) the user will be presented with a list of links for recovering the related objects first.

Note: In current implementation only required FKs are considered as a conflicts, but this would be changed in next versions since if relation existed for not required FK, and related object was deleted you will end up with `IntegrityError` when you will try to restore this object.

In such cases if you know which related object is missing and if it was registered with `aldryn-reversion` for revision tracking and with admin mixin - you can recover related object first and then return to restore this object.

If a related object was not registered with the admin mixin but was registered for revision tracking `aldryn-reversion` will attempt to resolve the conflict automatically by examining the required FK relations. If automatic conflict resolution was successful you will be able to restore the object and its required relations.

Note: Note that automatic conflict resolution will try to use the selected object revision. Related objects will be restored to the version at which they were in that revision - not the object's latest version. Be sure to examine the related objects carefully and edit them so that they are in the desired state.

Translations (django-parler)

When a model that is registered with `aldryn-reversion` contains translatable fields, recover form will also have the option to select translations to restore. In that case at least one translation should be selected.

CMS placeholder field

If a model has `placeholder` fields and `aldryn-reversion` was not configured to ignore those fields, they will also be tracked as part of object's revision.

In such cases, when the placeholder object representing the parent model's placeholder field is deleted, you will be notified and it will be restored as part of the recovery process.

Note: If only the placeholder object is deleted (and not the plugins that it holds) you will be able to restore both the placeholder and the plugins - in most cases the plugins will not have changed. If, on the other hand, both the placeholder and the related plugins were deleted, the result would be an empty placeholder. You may still be able to restore the plugins if, after recovering the deleted object, you restore an object version which contains the plugins.

When reverting history, the object will be restored to the corresponding revision automatically, so it may be a good idea to check the restored object and its plugins and edit them where necessary.

CHAPTER 3

Key topics

Limitations

Revert and recover

Be aware that reverting a model instance will also revert all related (i.e. via a foreign key) object instances to the same history point - this will not always be the desired behaviour, and may not be what is expected by your site's content editors.

Possible database integrity errors

If:

- a model has non-nullable foreign keys (`null=False`), *and*
- these foreign keys are not registered using the *follow* option, *and*
- a reversion action deletes a related object

then you will face a database `IntegrityError`.

To avoid this, ensure that such foreign keys are registered using the the *follow* keyword.

Missing admin options

If the **Recover deleted** button on a model's changelist admin view seems to be missing, or you cannot access the *history revert* mechanism, then most likely this model has not been correctly registered with `VersionedPlaceholderAdminMixin`. See *Admin registration*