

---

# **django-adminjournal**

**Aug 30, 2019**



---

## Contents

---

<b>1</b>	<b>Features</b>	<b>3</b>
<b>2</b>	<b>Requirements</b>	<b>5</b>
<b>3</b>	<b>Prepare for development</b>	<b>7</b>
<b>4</b>	<b>Resources</b>	<b>9</b>
4.1	Installation . . . . .	9
4.2	Usage . . . . .	10
4.3	Changelog . . . . .	10
4.4	API Reference . . . . .	10
<b>5</b>	<b>Indices and tables</b>	<b>17</b>
	<b>Python Module Index</b>	<b>19</b>
	<b>Index</b>	<b>21</b>



This library added extended capabilities to log access to Django ModelAdmins.



# CHAPTER 1

---

## Features

---

- Log additions, changes, deletions of models via the Django admin
- Log read access to change lists and model instances (unsaved change views)
- Log calls to actions in changelists of ModelAdmins





## CHAPTER 2

---

### Requirements

---

django-adminjournal supports Python 3 only and requires at least Django 1.11. The package uses Django's JSONField. Therefore, PostgreSQL database backend is required.



## CHAPTER 3

---

### Prepare for development

---

A Python 3.6 interpreter is required in addition to pipenv.

```
$ pipenv install --python 3.6 --dev
$ pipenv shell
$ pip install -e .
```

Now you're ready to run the tests:

```
$ pipenv run py.test
```



- [Documentation](#)
- [Bug Tracker](#)
- [Code](#)

Contents:

## 4.1 Installation

- Install with pip:

```
pip install django-adminjournal
```

- Your `INSTALLED_APPS` setting:

```
INSTALLED_APPS = (  
    # ...  
    'adminjournal',  
)
```

### 4.1.1 Configuration options

- `ADMINJOURNAL_PERSISTENCE_BACKEND` defines the backend that is used to store/persist the journal entries. Default is a database backend.
- `ADMINJOURNAL_MODEL_WHITELIST` defines the models to automatically activate the `ModelAdmin` mixin. The settings should be a list of Django models (e.g. `auth.User`) or the string `'__all__'` to activate the admin journal for all models.
- `ADMINJOURNAL_ENTRY_EXPIRY_DAYS` defines the number of days after which the journal entries are deleted when calling the management command `clearadminjournal`. The default is 365 days.

## 4.2 Usage

After adding `adminjournal` to `INSTALLED_APPS`, the journal is activated for all model admins added to Django's default `AdminSite` (`django.contrib.admin.site`).

### 4.2.1 Cleanup

Journal entries can be deleted automatically after a given amount of time (see configuration option `ADMINJOURNAL_ENTRY_EXPIRY_DAYS`). To do so, run the `clearadminjournal` regularly using a cron daemon or some other trigger tool.

If you use `uwsgi`, you might even use the built in cron helper:

```
[uwsgi]
# ...

# Django session cleanup
cron = 30 4 -1 -1 -1 django-admin clearsessions

# Adminjournal cleanup
cron = 15 4 -1 -1 -1 django-admin clearadminjournal
```

This would run the cleanup command every day at 4:15 am.

## 4.3 Changelog

### 4.3.1 0.1.0 (2018-11-16)

- Add management command to clean up old journal entries

### 4.3.2 0.0.1 (2018-11-12)

- Initial release of *django-adminjournal*

Api documentation:

## 4.4 API Reference

### 4.4.1 adminjournal package

#### Subpackages

#### `adminjournal.persistence_backends` package

#### Submodules

## adminjournal.persistence\_backends.base module

**class** adminjournal.persistence\_backends.base.**BaseBackend**

Bases: `object`

Base backend to persist journal entries.

Every backend must provide at least a *persist* method.

**persist** (*entry*)

The *persist* method is able to persist instances of *adminjournal.entry.Entry* classes. The method will return *True* or *False* to signal success.

## adminjournal.persistence\_backends.db module

**class** adminjournal.persistence\_backends.db.**Backend**

Bases: *adminjournal.persistence\_backends.base.BaseBackend*

Database-backed persistence layer for journal entries. Uses *adminjournal.Entry* model to store entries to database.

**persist** (*entry*)

The *persist* method is able to persist instances of *adminjournal.entry.Entry* classes. The method will return *True* or *False* to signal success.

## adminjournal.persistence\_backends.log module

**class** adminjournal.persistence\_backends.log.**Backend**

Bases: *adminjournal.persistence\_backends.base.BaseBackend*

Simple backend that uses Python-logging to “persist” a given entry.

**The log backend has two settings:**

- `ADMINJOURNAL_BACKEND_LOG_LOGGER`: Name of the logger to use
- `ADMINJOURNAL_BACKEND_LOG_LEVEL`: Valid logging level name to use

**persist** (*entry*)

The *persist* method is able to persist instances of *adminjournal.entry.Entry* classes. The method will return *True* or *False* to signal success.

## Submodules

### adminjournal.admin module

**class** adminjournal.admin.**EntryAdmin** (*model, admin\_site*)

Bases: `django.contrib.admin.options.ModelAdmin`

`search_fields = ('user_repr',)`

`list_display = ('__str__', 'action', 'content_type_repr', 'user_repr', 'object_id', '1`

`list_filter = ('action', 'content_type__app_label', 'user_repr')`

`date_hierarchy = 'timestamp'`

`readonly_fields = ('timestamp', 'action', 'user', 'user_repr', 'content_type', 'content`

**has\_add\_permission** (*request*, *obj=None*)

*has\_add\_permission* is overwritten to ensure no entries can be added.

**has\_delete\_permission** (*request*, *obj=None*)

*has\_delete\_permission* is overwritten to ensure nobody can remove entries.

**lazy\_description** (*obj*)

Helper to return the human readable entry description. If no description is available, the payload will be returned.

**object\_repr** (*obj*)

Helper to get the str-representation of the logged object.

**media**

### adminjournal.apps module

**class** adminjournal.apps.**AdminjournalConfig** (*app\_name*, *app\_module*)

Bases: django.apps.config.AppConfig

**name** = 'adminjournal'

**ready** ()

When loading the adminjournal app, we patch the Django admin site to ensure every model admin is hooked to the admin journal mixin if the setting ADMINJOURNAL\_PATCH\_ADMINSITE is set to True (default).

### adminjournal.entry module

**class** adminjournal.entry.**Entry** (*action*, *user*, *model\_class=None*, *model=None*, *description=None*, *timestamp=None*, *payload=None*)

Bases: object

This class represents a journal entry and provides methods to get information about the action which was tracked.

**ACTION\_VIEW** = 'view'

**ACTION\_ADD** = 'add'

**ACTION\_CHANGE** = 'change'

**ACTION\_DELETE** = 'delete'

**\_\_init\_\_** (*action*, *user*, *model\_class=None*, *model=None*, *description=None*, *timestamp=None*, *payload=None*)

Create a new entry instance.

The constructor handles the provided data and does some basis validation.

The parameters *action* and *user* are always required. It is possible to override the timestamp which used for that event.

In addition, *model\_class* and/or *model* needs to be provided. If both are given, the constructor will ensure that the *model\_class* fits the *provided model*.

**It is allowed to provide one of the following as *model\_class*:**

- Python class of a Django model
- ContentType model instance



- None (if none is provided, the `model_class` will be derived from the given model).

You don't have to provide a *model* if you already have the *model\_class* on hand.

The parameter *description* is useful to provide a human-readable representation of what happened.

**timestamp = None**

Point in time when the event happend.

**payload = None**

Dict-like object holding any other information related to the event.

**user\_repr**

Returns a human readable version of the user object.

**content\_type\_repr**

Returns a human readable version of the content type object.

**object\_id**

**persist ()**

Triggers the persisting of the instance.

## adminjournal.mixins module

**class** adminjournal.mixins.**JournalModelAdminMixin**

Bases: `object`

Mixin for ModelAdmin classes to issue journal entries on various actions via the model admin.

### Tracked actions:

- View changelist (w/ and w/o filters)
- View object change view
- Change object
- Add object
- Delete object
- Changelist actions (selected action and selected objects)

**log\_to\_adminjournal** (*action, user, message, model=None, payload=None*)

The `log_to_adminjournal` method requires at least the action type and the issuing user together with a human readable message or a `change_message`-style list from Django's `LogEntry`.

The method might use `change_message`-style lists to generate a human readable version of the data.

If a `change_message`-style input is provided, the payload is ignored.

If a str message is provided and the payload is a dictionary, the data is passed to the persistence layer.

**log\_addition** (*request, model, message*)

In addition to the Django `LogEntry`, add another entry to the adminjournal.

**log\_change** (*request, model, message*)

In addition to the Django `LogEntry`, add another entry to the adminjournal.

**log\_deletion** (*request, model, object\_repr*)

In addition to the Django `LogEntry`, add another entry to the adminjournal.

### **render\_change\_form** (*request, \*args, \*\*kwargs*)

If an object change view is requested (GET request on change view), an ACTION\_VIEW entry is generated to track read access to single objects.

### **response\_action** (*request, queryset*)

Actions on change lists are tracked too. To achieve this, this method parses the POST request and checks for various action fields to

- Learn what action was triggered
- What objects are included in the action call
- Are all objects involved or just some selected ones

We don't know what the action does, therefore all actions are tracked as ACTION\_VIEW.

### **changelist\_view** (*request, \*args, \*\*kwargs*)

GET requests on the changelist are tracked as ACTION\_VIEW entries. The changelist view might also return redirects. This case is handled by ensuring that a proper TemplateResponse is available and the "cl" context (which is the ChangeList instance) is present.

Parameters passed to the change list are tracked too. This allows to reconstruct the subset of objects a user viewed.

## adminjournal.models module

```
class adminjournal.models.Entry (id, timestamp, action, user, user_repr, content_type, content_type_repr, object_id, description, payload)
```

```
Bases: django.db.models.base.Model
```

### **timestamp**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

### **action**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

### **user**

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):  
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

### **user\_repr**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

### **content\_type**

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

#### **content\_type\_repr**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

#### **object\_id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

#### **description**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

#### **payload**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

#### **exception DoesNotExist**

Bases: `django.core.exceptions.ObjectDoesNotExist`

#### **exception MultipleObjectsReturned**

Bases: `django.core.exceptions.MultipleObjectsReturned`

#### **content\_type\_id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

```
get_next_by_timestamp(*, field=<django.db.models.fields.DateTimeField: timestamp>,
                      is_next=True, **kwargs)
```

```
get_previous_by_timestamp(*, field=<django.db.models.fields.DateTimeField: timestamp>,
                          is_next=False, **kwargs)
```

#### **id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

```
objects = <django.db.models.manager.Manager object>
```

#### **user\_id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

## adminjournal.monkeypatch module

```
adminjournal.monkeypatch.patch_admin_site(site)
```

This helper patches the default Django admin site to ensure the `JournalModelAdminMixin` is added to the model admins.

After patching the admin site, this helper checks all already registered model admins to be adminjournal enabled.

### adminjournal.persistence module

`adminjournal.persistence.persist` (*entry*, *backend=None*)

The *persist* function is the abstract endpoint to persist journal entries. The method receives a *adminjournal.entry.Entry* instance and an optional *backend* parameter to override the default persistence backend.

The return value is either *True* or *False*, to signal if the entry was saved.

`adminjournal.persistence.get_persistence_backend` (*path=None*)

Load a persistence backend and return a instance. If a path is provided, the backend is imported from that path. By default, `adminjournal.persistence_backends.db.Backend` is used.

## CHAPTER 5

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



**a**

adminjournal, 10  
adminjournal.admin, 11  
adminjournal.apps, 12  
adminjournal.entry, 12  
adminjournal.mixins, 13  
adminjournal.models, 14  
adminjournal.monkeypatch, 15  
adminjournal.persistence, 16  
adminjournal.persistence\_backends, 10  
adminjournal.persistence\_backends.base,  
    11  
adminjournal.persistence\_backends.db,  
    11  
adminjournal.persistence\_backends.log,  
    11





## Symbols

`__init__()` (*adminjournal.entry.Entry* method), 12

### A

`action` (*adminjournal.models.Entry* attribute), 14

`ACTION_ADD` (*adminjournal.entry.Entry* attribute), 12

`ACTION_CHANGE` (*adminjournal.entry.Entry* attribute), 12

`ACTION_DELETE` (*adminjournal.entry.Entry* attribute), 12

`ACTION_VIEW` (*adminjournal.entry.Entry* attribute), 12

`adminjournal` (*module*), 10

`adminjournal.admin` (*module*), 11

`adminjournal.apps` (*module*), 12

`adminjournal.entry` (*module*), 12

`adminjournal.mixins` (*module*), 13

`adminjournal.models` (*module*), 14

`adminjournal.monkeypatch` (*module*), 15

`adminjournal.persistence` (*module*), 16

`adminjournal.persistence_backends` (*module*), 10

`adminjournal.persistence_backends.base` (*module*), 11

`adminjournal.persistence_backends.db` (*module*), 11

`adminjournal.persistence_backends.log` (*module*), 11

`AdminjournalConfig` (*class in adminjournal.apps*), 12

### B

`Backend` (*class in adminjournal.persistence\_backends.db*), 11

`Backend` (*class in adminjournal.persistence\_backends.log*), 11

`BaseBackend` (*class in adminjournal.persistence\_backends.base*), 11

### C

`changelist_view()` (*adminjournal*

*nal.mixins.JournaledModelAdminMixin* method), 14

`content_type` (*adminjournal.models.Entry* attribute), 14

`content_type_id` (*adminjournal.models.Entry* attribute), 15

`content_type_repr` (*adminjournal.entry.Entry* attribute), 13

`content_type_repr` (*adminjournal.models.Entry* attribute), 15

### D

`date_hierarchy` (*adminjournal.admin.EntryAdmin* attribute), 11

`description` (*adminjournal.models.Entry* attribute), 15

### E

`Entry` (*class in adminjournal.entry*), 12

`Entry` (*class in adminjournal.models*), 14

`Entry.DoesNotExist`, 15

`Entry.MultipleObjectsReturned`, 15

`EntryAdmin` (*class in adminjournal.admin*), 11

### G

`get_next_by_timestamp()` (*adminjournal.models.Entry* method), 15

`get_persistence_backend()` (*in module adminjournal.persistence*), 16

`get_previous_by_timestamp()` (*adminjournal.models.Entry* method), 15

### H

`has_add_permission()` (*adminjournal.admin.EntryAdmin* method), 11

`has_delete_permission()` (*adminjournal.admin.EntryAdmin* method), 12

### I

`id` (*adminjournal.models.Entry* attribute), 15

## J

JournalModelAdminMixin (class in *adminjournal.mixins*), 13

## L

lazy\_description() (adminjournal.admin.EntryAdmin method), 12

list\_display (adminjournal.admin.EntryAdmin attribute), 11

list\_filter (adminjournal.admin.EntryAdmin attribute), 11

log\_addition() (adminjournal.mixins.JournalModelAdminMixin method), 13

log\_change() (adminjournal.mixins.JournalModelAdminMixin method), 13

log\_deletion() (adminjournal.mixins.JournalModelAdminMixin method), 13

log\_to\_adminjournal() (adminjournal.mixins.JournalModelAdminMixin method), 13

## M

media (adminjournal.admin.EntryAdmin attribute), 12

## N

name (adminjournal.apps.AdminJournalConfig attribute), 12

## O

object\_id (adminjournal.entry.Entry attribute), 13

object\_id (adminjournal.models.Entry attribute), 15

object\_repr() (adminjournal.admin.EntryAdmin method), 12

objects (adminjournal.models.Entry attribute), 15

## P

patch\_admin\_site() (in module *adminjournal.monkeypatch*), 15

payload (adminjournal.entry.Entry attribute), 13

payload (adminjournal.models.Entry attribute), 15

persist() (adminjournal.entry.Entry method), 13

persist() (adminjournal.persistence\_backends.base.BaseBackend method), 11

persist() (adminjournal.persistence\_backends.db.Backend method), 11

persist() (adminjournal.persistence\_backends.log.Backend method), 11

persist() (in module *adminjournal.persistence*), 16

## R

readonly\_fields (adminjournal.admin.EntryAdmin attribute), 11

ready() (adminjournal.apps.AdminJournalConfig method), 12

render\_change\_form() (adminjournal.mixins.JournalModelAdminMixin method), 13

response\_action() (adminjournal.mixins.JournalModelAdminMixin method), 14

## S

search\_fields (adminjournal.admin.EntryAdmin attribute), 11

## T

timestamp (adminjournal.entry.Entry attribute), 13

timestamp (adminjournal.models.Entry attribute), 14

## U

user (adminjournal.models.Entry attribute), 14

user\_id (adminjournal.models.Entry attribute), 15

user\_repr (adminjournal.entry.Entry attribute), 13

user\_repr (adminjournal.models.Entry attribute), 14