Django Admin Actions Documentation

Release 1.5

Stefano Apostolico

Jun 12, 2017

Contents

Collection of useful actions to use with django.contrib.admin.ModelAdmin and/or django.contrib.admin.AdminSite

CHAPTER 1

Table Of Contents

Installation

Installing django-adminactions is as simple as checking out the source and adding it to your project or `PYTHONPATH`.

1. First of all follow the instruction to install django_admin application,
2. Either check out django-adminactions from GitHub or to pull a release off PyPI. Doing `pip install django-adminactions` or `easy_install django-adminactions` is all that should be required.
3. Either symlink the `adminactions` directory into your project or copy the directory in. What ever works best for you.

Install test dependencies

If you wanto to run *adminactions* tests you need extra requirements

```
pip install -U django-adminactions[tests]
```

Configuration

Add *adminactions* to your *INSTALLED_APPS*:

```
INSTALLED_APPS = (
    'adminactions',
    'django.contrib.admin',
    'django.contrib.messages',
)
```

Add the actions to your site:

```
from django.contrib.admin import site
import adminactions.actions as actions

# register all adminactions
actions.add_to_site(site)
```

Add service url to your urls.py

```
urlpatterns = patterns('',
    ...
    (r'^adminactions/', include('adminactions.urls')),
)
```

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:

Types of Contributions

Report Bugs

Report bugs at https://github.com/saxix/django-adminactions/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.

Fix Bugs

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

Implement Features

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

Write Documentation

django-adminactions could always use more documentation, whether as part of the official django-adminactions docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/saxix/django-adminactions/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 :)

Get Started!

Ready to contribute? Here's how to set up *django-adminactions* for local development.

1. Fork the *django-adminactions* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/django-adminactions.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-adminactions
$ cd django-adminactions/
$ pip install -e .[dev]
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

5. Install your preferred Django version

    $ pip install django

Now you can make your changes locally.

6. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ make qa
$ tox
```

7. 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.

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 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/saxix/django-adminactions/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ py.test tests/<FILE>
```

Actions

Mass Update

Update one or more fields of the selected queryset to a common value and/or apply a *transform_operations* to selected field.

validate	use obj.save() instead of obj._default_manager.update. Slower but required in some cases (To run some business logic in save() and clean() Manadatory if use *Transform Operation*
unique_transaction	`Please see the changelog`

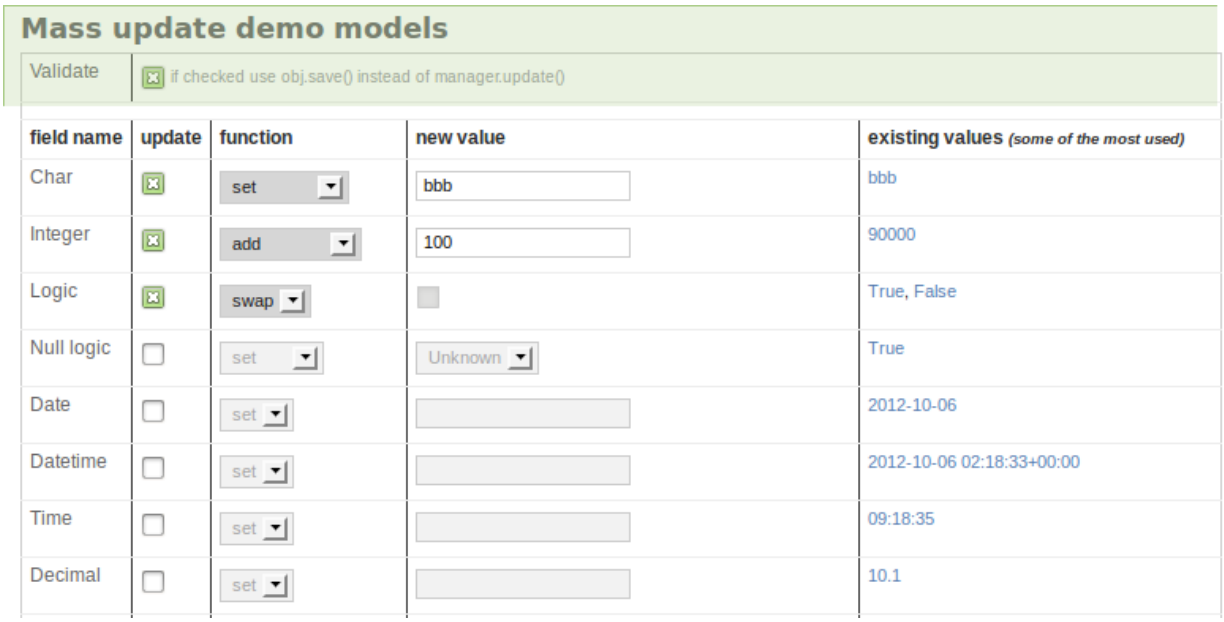
Screenshot

Transform Operation

`Please see the changelog` Is possible to update fields applying function. django-adminactions comes with a predefined set of functions. You can anyway *register your own functions*

common to all models

set	set the value
set null	set the value to null (only available if the field has null=True



`django.db.models.CharField`

upper	convert to uppercase
lower	convert to lowercase
capitalize	capitalize first character
capwords	capitalize each word
swapcase	swap the case
trim	remove leading and trailing whitespace

`django.db.models.IntegerField`

add	add the value passed as arg to the current value of the field
sub	subtract the value passed as arg to the current value of the field
add_percent	add the *X* percent to the current value
sub_percent	subtract the *X* percent from the current value

`django.db.models.BooleanField`

swap	invert (negate) the current value

`django.db.models.NullBooleanField`

swap	invert (negate) the current value

`django.db.models.EmailField`

change domain	set the domain (@?????) to common value

`django.db.models.URLField`

change domain	set the protocol (????://) to common value

Export as CSV

Export selected queryset as csv file. (see csv)

Available options: (see Dialects and Formatting Parameters).

See also:

Are you looking for the *export_as_csv* internal api? .

header	add the header line to the file
delimiter	A one-character string used to separate fields. It defaults to ‘,’. (see `csv.Dialect.delimiter`)
quotechar	A one-character string used to quote fields containing special characters, such as the delimiter or quotechar, or which contain new-line characters. It defaults to ‘”’. (see `csv.Dialect.quotechar`)
quoting	Controls when quotes should be generated by the writer and recognised by the reader. (see `csv.Dialect.quoting`)
escapechar	A one-character string used by the writer to escape the *delimiter*. (see `csv.Dialect.escapechar`)
datetime_format	How to format datetime field. (see strftime and strptime Behavior)
date_format	How to format date field. (see strftime and strptime Behavior)
time_format	How to format time field. (see strftime and strptime Behavior)
columns	Which columns will be included in the dump

Screenshot

Streaming CSV Response

When a very large/complex dataset is exported, it may take be useful to stream the response row by row instead of the whole file.

To enable this functionality by default, set the django `settings.ADMINACTIONS_STREAM_CSV` to `True` (default: `False`).

Behind the scenes, this attaches an iterator as the response content (using a `StreamingHttpResponse` *if django >= 1.6* and `HttpResponse` otherwise); the iterator simply yields a new csv row for each queryset iteration.

The benefit of this approach is a shorter initial response which unblocks the customer from request/response and he is free to do other things while waiting for the download to finish.

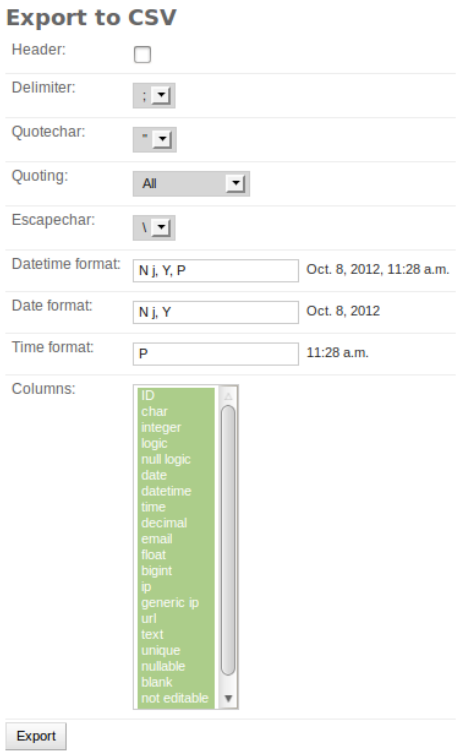
See also:

‘csv_defaults‘_

Export as Fixture

Export selected queryset as fixtures using any registered Serializer.

Note: this is not the same as django’s command `dumpdata` because it can dump also the Foreign Keys.

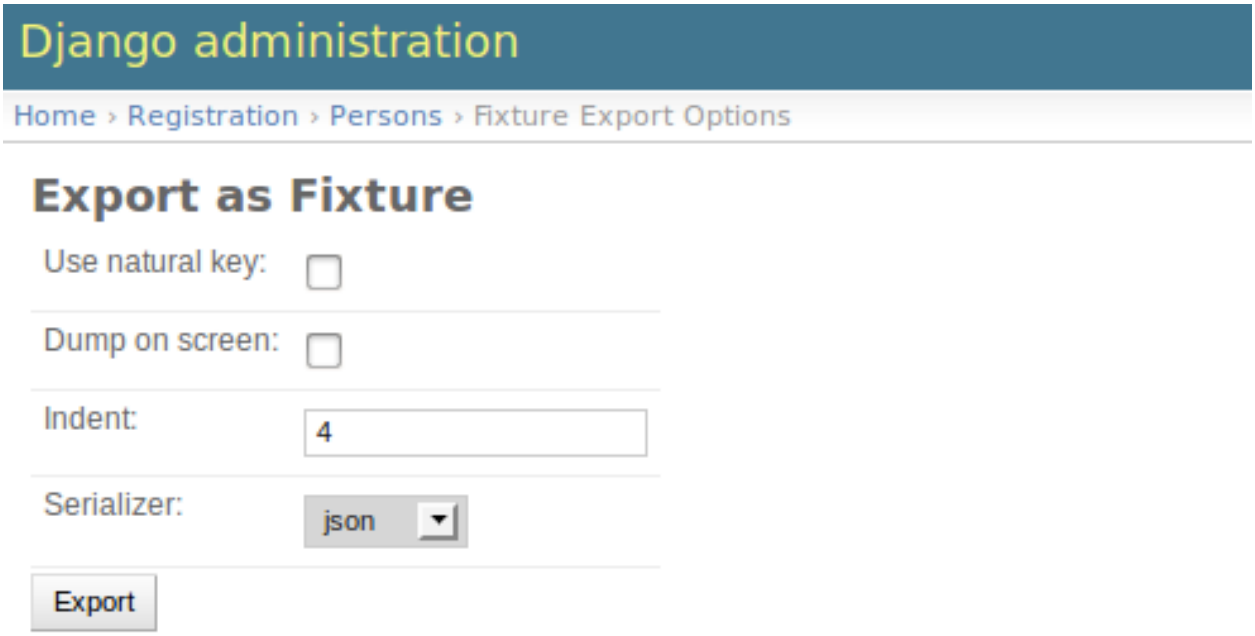


The following characters are recognized in the *format* parameter string

format character	Description	Example returned values
Day	---	---
d	Day of the month, 2 digits with leading zeros	*01* to *31*
D	A textual representation of a day, three letters	*Mon* through *Sun*
j	Day of the month without leading zeros	*1* to *31*
l (lowercase 'L')	A full textual representation of the day of the week	*Sunday* through *Saturday*
1 (for Monday) through *7* (for Sunday)		
S	English ordinal suffix for the day of the month, 2 characters	*st*, *nd*, *rd* or *th*. Works well with *j*
w	Numeric representation of the day of the week	*0* (for Sunday) through *6* (for Saturday)
z	The day of the year (starting from 0)	*0* through *365*
Week	---	---
W	ISO-8601 week number of year, weeks starting on Monday (added in PHP 4.1.0)	Example: *42* (the 42nd week in the year)
Month	---	---
F	A full textual representation of a month, such as January or March	*January* through *December*
m	Numeric representation of a month, with leading zeros	*01* through *12*
M	A short textual representation of a month, three letters	*Jan* through *Dec*
n	Numeric representation of a month, without leading zeros	*1* through *12*
t	Number of days in the given month	*28* through *31*
Year	---	---
L	Whether it's a leap year	*1* if it is a leap year, *0* otherwise.
Examples: *1999* or *2003*		
Y	A full numeric representation of a year, 4 digits	Examples: *1999* or *2003*
y	A two digit representation of a year	Examples: *99* or *03*
Time	---	---

use natural key	If true use natural keys.
dump on screen	Dump on screen instead to show `Save as` popup
indent	Indentation value
serializer	Serializer to use. (see Serialization formats)
add_foreign_keys	If checked export foreign keys too, otherwise act as standard dumpdata

Screenshot



Export Delete Tree

`Please see the changelog` Export all the records that belong selected queryset using any registered Serializer.

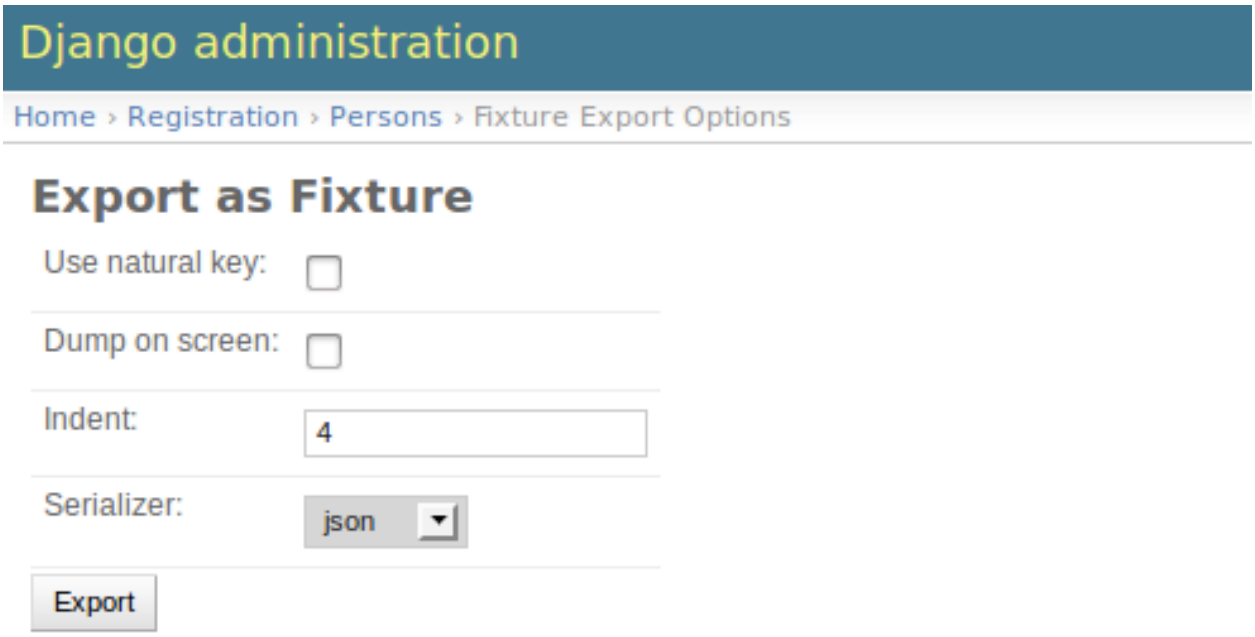
This action is the counterpart of *export_as_fixture*, where it dumps the queryset and it's ForeignKeys, *export_delete_tree* all the records that belong to the entries of the selected queryset. see *export_as_fixture* for details

use natural key	If true use natural keys.
dump on screen	Dump on screen instead to show `Save as` popup
indent	Indentation value
serializer	Serializer to use. (see Serialization formats)
add_foreign_keys	If checked export dependent objects too.

Screenshot

Export as Excel

`Please see the changelog` Export selected queryset as Excel (xls) file.



Available options:

header	add the header line to the file
columns	Which columns will be included in the dump

Graph Queryset

Graph selected queryset.

Graph type	Graph type to use
Group by and count by:	Grouping field

Screenshot

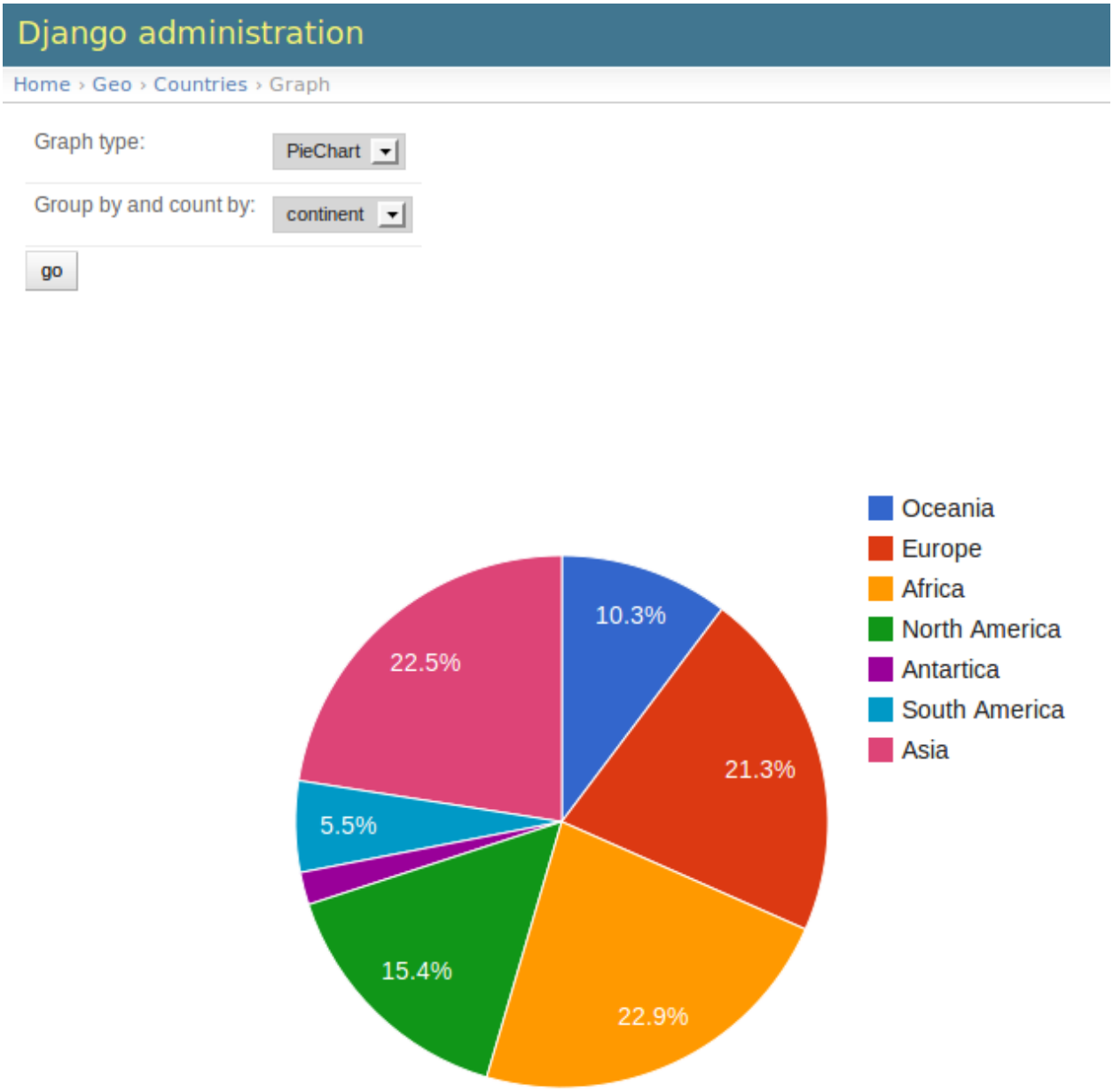
Screenshot

Merge Records

`Please see the changelog` Sometimes you need to *"merge"* two records maybe because they represent the same business object but were create double by mistake. This action allow you to selectively merge two records and move dependencies from one record to the other one.

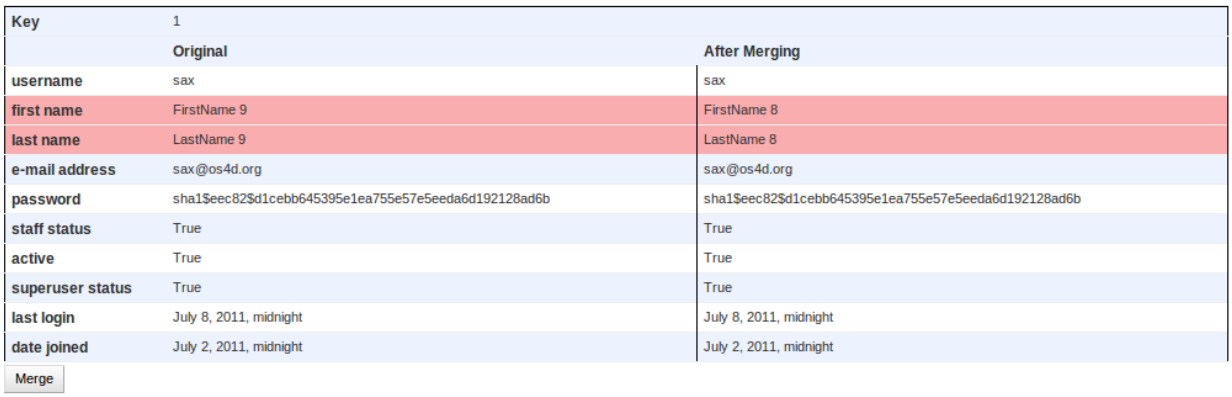
Screenshots

Step 1


Django administration
Home › Geo › Countries › Graph
Graph type:
PieChart
Group by and count by:
continent
go
10.3%
21.3%
22.9%
15.4%
5.5%
22.5%
Oceania
Europe
Africa
North America
Antartica
South America
Asia


Django administration
Home › Geo › Countries › Graph
Graph type:
BarChart
Group by and count by:
continent
go
Slices
Oceania
Europe
Africa
North America
Antartica
South America
Asia
0
15
30
45
60
Dependencies
Move
Search GenericForeignKeys
No
Field
Master #1 (This will be preserved)
swap
Result
Other #10 (This will be removed)
username
sax
sax
user_8
first name
FirstName 9
FirstName 8
FirstName 8
last name
LastName 9
LastName 8
LastName 8
e-mail address
sax@os4d.org
sax@os4d.org
user_8@test.org
password
sha1$eec82$d1cebb645395e1ea755e57e5eeda6d192128ad6b
sha1$eec82$d1cebb645395e1ea755e57e5eeda6d192128ad6b
staff status
True
True
False
active
True
True
True
superuser status
True
True
False
last login
July 8, 2011, midnight
July 8, 2011, midnight
July 8, 2011, midnight
date joined
July 2, 2011, midnight
July 2, 2011, midnight
July 2, 2011, midnight
Preview

Step 2



Key	1	
	Original	After Merging
username	sax	sax
first name	FirstName 9	FirstName 8
last name	LastName 9	LastName 8
e-mail address	sax@os4d.org	sax@os4d.org
password	sha1$eec82$d1cebb645395e1ea755e57e5eeda6d192128ad6b	sha1$eec82$d1cebb645395e1ea755e57e5eeda6d192128ad6b
staff status	True	True
active	True	True
superuser status	True	True
last login	July 8, 2011, midnight	July 8, 2011, midnight
date joined	July 2, 2011, midnight	July 2, 2011, midnight

Merge

Limitations/TODO

- merge doesn't work for models related with `on_delete=Protect` (see #85)

Signals

django-adminactions provides the following signals:

- *adminaction_requested*
- *adminaction_start*
- *adminaction_end*

adminaction_requested

Sent when the action is requested (ie click 'Go' in the admin changelist view). The handler can raise a *ActionInterrupted* to interrupt the action's execution. The handler can rely on the following parameter:

- sender: `django.db.models.Model`
- action: string. name of the action
- request: `django.core.httpd.HttpRequest`
- queryset: `django.db.models.query.Queryset`
- modeladmin: `django.contrib.admin.ModelAdmin`

Example:

```
from adminactions.signals import adminaction_requested

def myhandler(sender, action, request, queryset, modeladmin, **kwargs):
    if action == 'mass_update' and queryset.filter(locked==True).exists():
        raise ActionInterrupted('Queryset cannot contains locked records')

adminaction_requested.connect(myhandler, sender=MyModel, action='mass_update`)
```

adminaction_start

Sent after the form has been validated (ie click 'Apply' in the action Form), **just before** the execution of the action. The handler can raise a *ActionInterrupted* to avoid the stop execution. The handler can rely on the following parameter:

- sender: `django.db.models.Model`
- action: string. name of the action
- request: `django.core.httpd.HttpRequest`
- queryset: `django.db.models.query.Queryset`
- modeladmin: `django.contrib.admin.ModelAdmin`
- form: `django.forms.Form`

Example:

```
from adminactions.signals import adminaction_start

def myhandler(sender, action, request, queryset, modeladmin, form, **kwargs):
    if action == 'export' and 'money' in form.cleaned_data['columns']:
        if not request.user.is_administrator:
            raise ActionInterrupted('Only administrors can export `money` field')

adminaction_start.connect(myhandler, sender=MyModel, action='export`)
```

adminaction_end

Sent **after** the successfully execution of the action. The handler can rely on the following parameter:

- sender: `django.db.models.Model`
- action: string. name of the action
- request: `django.core.httpd.HttpRequest`
- queryset: `django.db.models.query.Queryset`
- modeladmin: `django.contrib.admin.ModelAdmin`
- form: `django.forms.Form`
- errors: dict
- updated: int

Exceptions

ActionInterrupted

Exception raised to interrupt an action.

API

Functions

export_as_csv

See also:

Are you looking for the *Export as CSV* action? .

adminactions.api.**export_as_csv**()

Exports a queryset as csv from a queryset with the given fields. **Defaults**

> **Warning:** Due a mistake the default configuration of *export_as_csv* is not *csv* but *semicolon-csv*

```
csv_options_default = {'date_format': 'd/m/Y',
                       'datetime_format': 'N j, Y, P',
                       'time_format': 'P',
                       'header': False,
                       'quotechar': '"',
                       'quoting': csv.QUOTE_ALL,
                       'delimiter': ';',
                       'escapechar': '\\', }
```

Usage examples

Returns `HttpResponse`:

```
response = export_as_csv(User.objects.all())
```

Write to file

```
>>> users = export_as_csv(User.objects.all(), out=open('users.csv', 'w'))
>>> users.close()
```

Write to buffer

```
>>> users = export_as_csv(User.objects.all(), out=StringIO())
>>> with open('users.csv', 'w') as f:
        f.write(users.getvalue())
```

Export with callable

```
>>> fields = ['username', 'get_full_name']
>>> export_as_csv(User.objects.all(), fields=fields, out=sys.stdout)
"sax";"FirstName 9 LastName 9"
"user_0";"FirstName 0 LastName 0"
"user_1";"FirstName 1 LastName 1"
```

Export with dictionaries

```
>>> fields = ['codename', 'content_type__app_label']
>>> qs = Permission.objects.filter(codename='add_user').values('codename', 'content_
↪type__app_label')
```

```
>>> __ = export_as_csv(qs, fields=fields, out=sys.stdout)
"add_user";"auth"
```

export_as_xls

`Please see the changelog` Exports a queryset as csv from a queryset with the given fields.

merge

See also:

See *Merge Records* action for additional notes.

Merge 'other' into master.

> `fields` is a list of fieldnames that must be read from `other` to put into master. If `fields` is None `master` will get all the `other` values except primary_key. Finally `other` will be deleted and master will be preserved

Custom validation

If you need to disable validation for some fields, it is possible to set parameter `merge_form` to a subclass of *:class:adminactions.merge.MergeForm* and change the validation there.

```
class CompanyMergeForm(merge.MergeForm):
    class Meta:
        model = models.Company
        fields = "__all__"

    def full_clean(self):
        super().full_clean()
        if 'address_psc' in self._errors:
            del self._errors['address_psc']

class CompanyAdmin(city_admin_mixin_generator(admin.ModelAdmin)):
    form = CompanyForm
    merge_form = CompanyMergeForm
```

Filename callbacks

To use custom names for yours exports simply implements `get_export_<TYPE>_filename` in your `Modeladmin` class, these must return a string that will be used as filename in the SaveAs dialog box of the browser

example:

```
class UserAdmin(ModelAdmin):
    def get_export_as_csv_filename(request, queryset):
        if 'isadmin' in request.GET
            return 'administrators.csv'
        else:
            return 'all_users.csv'
```

Available callbacks:

- get_export_as_csv_filename
- get_export_as_fixture_filename
- get_export_delete_tree_filename

Utils

adminactions.utils.**clone_instance**(*instance*, *fieldnames=None*)

returns a copy of the passed instance.

Parameters **instance** – django.db.models.Model instance

Returns django.db.models.Model instance

adminactions.utils.**get_field_by_path**(*model*, *field_path*)

get a Model class or instance and a path to a attribute, returns the field object

Parameters

- **model** – django.db.models.Model
- **field_path** – string path to the field

Returns django.db.models.Field

```
>>> from django.contrib.auth.models import Permission
```

```
>>> p = Permission(name='perm')
>>> get_field_by_path(Permission, 'content_type').name
'content_type'
>>> p = Permission(name='perm')
>>> get_field_by_path(p, 'content_type.app_label').name
'app_label'
```

adminactions.utils.**get_field_value**(*obj*, *field*, *usedisplay=True*, *raw_callable=False*)

returns the field value or field representation if get_FIELD_display exists

Parameters

- **obj** – django.db.models.Model instance
- **field** – django.db.models.Field instance or basestring fieldname
- **usedisplay** – boolean if True return the get_FIELD_display() result

Returns field value

```
>>> from django.contrib.auth.models import Permission
>>> p = Permission(name='perm')
>>> get_field_value(p, 'name') == 'perm'
True
>>> get_field_value(p, None)
Traceback (most recent call last):
    ...
ValueError: Invalid value for parameter `field`: Should be a field name or a␣
↪Field instance
```

adminactions.utils.**get_verbose_name**(*model_or_queryset*, *field*)
returns the value of the verbose_name of a field

typically used in the templates where you can have a dynamic queryset

Parameters

- **model_or_queryset** (django.db.models.Model, django.db.query.Queryset) – target object
- **field** (django.db.models.Field, basestring) – field to get the verbose name

Returns translated field verbose name

Return type unicode

Valid uses:

```
>>> from django.contrib.auth.models import User, Permission
>>> user = User()
>>> p = Permission()
>>> get_verbose_name(user, 'username') == 'username'
True
>>> get_verbose_name(User, 'username') == 'username'
True
>>> get_verbose_name(User.objects.all(), 'username') == 'username'
True
>>> get_verbose_name(User.objects, 'username') == 'username'
True
>>> get_verbose_name(User.objects, user._meta.fields[0]) == 'ID'
True
>>> get_verbose_name(p, 'content_type.model') == 'python model class name'
True
```

Templatetags

adminactions.templatetags.actions.**verbose_name**(*model_or_queryset*, *field*)
templatetag wrapper to *adminactions.utils.get_verbose_name*

adminactions.templatetags.actions.**field_display**(*obj*, *field*)
returns the representation (value or get_FIELD_display()) of a field

see *adminactions.utils.get_field_value*

HowTo

- *Register Your Own Transform Function*
- *Customize Massupdate Form*
- *Selectively Register Actions*

Register Your Own Transform Function

Transform Operation are manage by proper tranform functions, django-adminactions come with a small set but it's possible to add extra function. Transform function are function that accept one or two parameter.

Customize Massupdate Form

`Please see the changelog` To customize the Form used by the massupdate action simply create your own Form class and set it as value of the `mass_update_form` attribute to your `ModelAdmin`. ie:

```
class MyMassUpdateForm(ModelForm):
    class Meta:
        model = MyModel


class MyModelAdmin(admin.ModelAdmin):
    mass_update_form = MyMassUpdateForm


admin.register(MyModel, MyModelAdmin)
```

Selectively Register Actions

To register only some selected action simply use the `site.add_action` method:

```
from django.contrib.admin import site
import adminactions.actions as actions


site.add_action(actions.mass_update)
site.add_action(actions.export_as_csv)
```

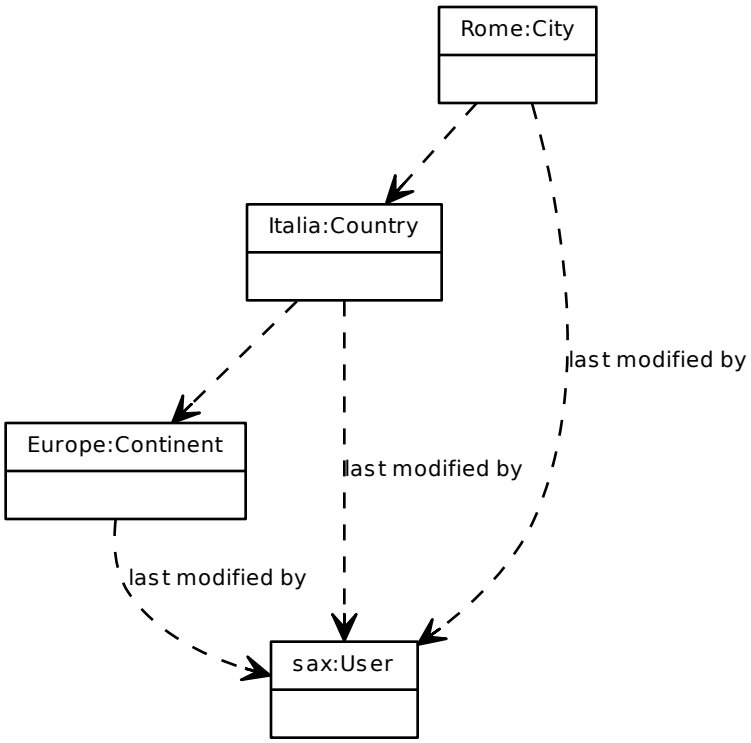
FAQ

It's possible to disable selenium tests?

Simply add `ENABLE_SELENIUM=False` to your `settings` file, or set DISABLE_SELENUIM environment variable

Difference between Export Delete Tree and Export as Fixture

Considering this:



here the partial code:

```
from django.contrib.auth.models import User

class Continent(models.Model):
    label = models.CharField(...)
    last_modified_by = models.ForeignKey(User)

class Country(models.Model):
    continent = models.ForeignKey(Continent)
    label = models.CharField(...)
    last_modified_by = models.ForeignKey(User)

class City(models.Model):
    country = models.ForeignKey(Country)
    label = models.CharField(...)
    last_modified_by = models.ForeignKey(User)
```

- Selecting sax (User) as target model:

  `export_delete_tree()` will dump the whole tree

  `export_as_fixture()` will dump only `sax`

- Selecting Rome (City) as target model:

  `export_delete_tree()` will dump only `Rome`

`export_as_fixture()` will dump the whole tree

CHAPTER 2

Project Info

Changelog

This sections lists the biggest changes done on each release.

Release 1.5

- add official support to Django 1.11 and Python 3.6
- fixes #116 Fixing ManyToMany merging with intermediary models. (thanks int-ua)
- fixes #95 Cannot merge models with subclassed ImageField or FileField: "file not sent". (thanks int-ua)
- fixes #108 merge doesn't account for many-to-many relationships
- fixes #93 Do not export dates as strings in Excel

Release 1.4

- document #112 Undocumented feature: merge_form
- document #108 merge doesn't account for many-to-many relationships
- document #95 Cannot merge models with subclassed ImageField: "file not sent" error
- document #85 merge doesn't work for models related with on_delete=Protect

Release 1.3 ==========fir * fixes #92 translations are not compiled in package * fixes #105 Support exporting many to many fields * fixes #109 AttributeError: module 'adminactions.compat' has no attribute 'nocommit'

Release 1.2

- merge #98 - Django 1.10 support (thanks PetrDlouhy, florianm)

Release 1.1

- merge #91 - add french translation
- merge #88 - Display required columns in byrows_update formset
- merge #87 - Add AdminSite.each_context() to templates context.
- merge #86 - Compilemessages failed for Spanish translation
- merge #83 - byrows_update action: adapt test_permissions to take into byrows update action.
- merge #79 - Permissions don't work

Release 1.0

- minor refactoring
- official support for django 1.4 to 1.9
- official support for python 2.7, 3.3, 3.5
- merge #77 - add initial form values for CSV export
- merge #76 - reuse xls style while iterating over cells
- merge #75 - allow streaming export to CSV response
- merge #73 - Fixing merge on DateTimeField with null=True
- support settings.ADMINACTIONS_CSV_OPTIONS_DEFAULT
- support streaming CSV file response
- new *upper* and *lower* modifiers available for EmailField in mass update.

Release 0.8.5

- repackage due broken version in 0.8.4

Release 0.8.4

- fixes #70 get_models return incorrect models in django 1.7+
- closes #71

Release 0.8.3

- bugfix: support both post_syncdb and post_migrate

Release 0.8.2

- fixes #64: Export not working when actions enabled on top & bottom
- document #62: default of csv is not csv (thanks @oppianmatt)

Release 0.8.1

- Use collections.OrderedDict instead for Django1.7 or higher. (thanks @rvoicilas)

Release 0.8

- python 3.3, 3.4 compatibility
- add spanish translation (thanks @xangmuve)

Release 0.7

- fixes issue in mass_update due wrong indentaion
- fixed #49
- removed options to enable/disable transactions during mass_update.
- fixed #60

Release 0.6

- fixed #55
- fixed #51
- added selenium tests
- pulled out tests from main package and use of py.test
- removed demoproject (use *make demo* instead)

Release 0.5

- fix mass_update bug that caused all records in a table to be updated (thanks @jht001)
- Added timezone support to csv and xls export

Release 0.4

- fixed #33
- fixed #20

Release 0.3

- fixed #26
- add feature to *use callable as columns*
- add feature to *export dictionaries*
- new action *Export as Excel*
- added custom headers to *Export as CSV*
- new permission adminactions_massupdate
- new permission adminactions_merge

Release 0.2

- improved *Export as CSV*
- Django 1.6 compatibility
- Added *modeladmin* in providing_args of signals: *adminaction_requested*, *adminaction_start*, *adminaction_end*

Release 0.1

- new api module
- pull out core export_csv functionalites as _export_as_csv() to be used by custom code
- New exported filename callback for easy customize the filename (see *Filename callbacks*)
- New registration shortcut add_to_site()
- New action: *Merge Records*
- Fixed #9
- Added permissions
- New signals: *adminaction_requested*, *adminaction_start*, *adminaction_end*

Release 0.0.4

- NEW added `add_foreign_keys` parameter to *Export as Fixture* and *Export Delete Tree*
- NEW *Export Delete Tree*
- **renamed export_as_json as export_as_fixtures**
  - added foreign_keys dumps
  - multiple serializer
- NEW: *Transform Operation*
- NEW: url to preview date format in *export_as_csv*

Release 0.0.3

- added demo project

Release 0.0.2

- name changed from django-actions to django-adminactions

Release 0.0.1

- first relase

Credits

Author

- Stefano Apostolico <s.apostolico@gmail.com>

Contributors

- tdruez (@tdruez)
- hanfeisun (@hanfeisun)
- Narsil (@Narsil)
- joelryan2k (@joelryan2k)
- hekevintran (@hekevintran)
- mrc75 (@mrc75)
- ipmb (@ipmb)
- Peter Baumgartner (@ipmb)
- BoscoMW (@Mbosco)
- Bertrand Bordage (@BertrandBordage)
- jht001 (@jht001)
- Cesar Abel (@xangmuve)
- Viator (@viatoriche)
- Serhiy Zahoriya (@int-ua)
- Pavel Savchenko (@asfaltboy)

CHAPTER 3

Links

- Project home page: https://github.com/saxix/django-adminactions
- Issue tracker: https://github.com/saxix/django-adminactions/issues?sort
- Download: http://pypi.python.org/pypi/django-adminactions/
- Documentation: http://readthedocs.org/docs/django-adminactions/en/latest/

CHAPTER 4

Indices and tables

- genindex
- search

Python Module Index

a

Index