
Django-data-exports Documentation

Release 0.1

Mathieu Agopian

May 16, 2017

Contents

1	Requirements	3
2	Installation	5
3	Usage	7
3.1	Using the admin	7
3.2	Using the included example views	7
3.3	Export columns	8
3.4	Getattribute filter	8
3.5	Nice_display filter	8
4	Advanced usage	11
4.1	Export formats	11
4.2	Using your own views	11
4.3	Using your own templates	12
5	Changes	13
6	Indices and tables	15

Django-data-exports is a model data exports app for Django. It allows you to easily create exports for your models.

Adding this app to your project will let you create exports for your models, and customize the data that will be exported by specifying which columns to include, and which format to use.

Typical use case: display a few columns from one of your models as a HTML table to be easily copy/pasted to a spreadsheet.

The source code is [available on Github](#) under the 3-clause BSD licence.

CHAPTER 1

Requirements

- Django `>= 1.3`: `django-data-exports` makes use of class based views
- `django-inspect-model`: this app is used to populate column names from the model to be exported

CHAPTER 2

Installation

```
pip install django-data-exports
```

Then add to your project's `INSTALLED_APPS`. In `settings.py`:

```
INSTALLED_APPS = (  
    '...',  
    # whatever you already have  
    '...',  
    'data_exports',  
)
```

Install the models:

```
./manage.py syncdb # or ./manage.py migrate if you're using south
```

And finally, plug the urls to your `ROOT_URLCONF`:

```
urlpatterns = patterns('',  
    # ... all the other urls you already have  
  
    # exports  
    url(r'^exports/', include('data_exports.urls', namespace='data_exports')),  
)
```


CHAPTER 3

Usage

Either add exports through the admin, or use the included example views. If there's no export format attached to an export, the `data_exports/export_detail.html` template will be rendered with the following context:

- `export`: the export itself
- `data`: a queryset of all the `export.model`'s instance

Using the admin

There's nothing specific to do here: connect to the admin, and add new exports. A few things to note:

- when you create an export, it's not possible to add columns at first. The reason being that the model is needed to be able to populate the column names
- when you add an export, clicking on the “save” button will have the same effect as clicking on “save and continue editing”
- once an export is created, and is being edited, the columns can be added (and are displayed as inlines)

Using the included example views

There's three included example views:

- `/exports/add`: create a new export
- `/exports/<export slug>/columns`: add columns to your export
- `/exports/<export slug>`: visualize your export

There is, at the moment, no example view for the export formats.

Export columns

Column choices make use of [django-inspect-model](#) to build the list of accessible “items”. Please check this app’s documentation to know more about “items”.

Choices are built by `data_exports.forms.get_choices`, and will consist of all the accessible items on the exported model, and on all its related models. The only related fields accessible are those on models that are directly related, using forward or reverse OneToOne fields and forward ForeignKey fields.

Example:

```
class Foo(models.Model):
    name = CharField(max_length=50)
    bar = ForeignKey(Bar)

class Bar(models.Model):
    name = CharField(max_length=50)
```

An export of `Foo` will have the following column choices:

- `name: Foo.name`
- `bar: Foo.bar`, which is `unicode(Foo.bar)`
- `bar.name: Bar.name`

To display the value of those columns, the included templates use `data_exports.template_tags.getter_tags`:

Getattribute filter

```
{% load getter_tags %}
{{ obj|getattribute:column }}
```

This is roughly equivalent to the `getattr` python builtin, but can cope with column choices:

- if `column` doesn’t have a dot, return `getattr(obj, column)`, or `getattr(obj, column)()` if it’s a callable
- if `column` does have a dots (eg: `bar.name`), recursively call `getattribute()` to get to the final attribute:

```
attr = getattribute(obj, 'bar.name')
# equivalent to:
temp = getattr(obj, 'bar')
attr = getattr(temp, 'name')
```

Nice_display filter

```
{% load getter_tags %}
{{ obj|getattribute:column|nice_display }}
```

For now, all this does is return a comma-separated list of related instances for a many-to-many field.

If the `item` field has an `all` method:

```
return ', '.join(map(unicode, item.all()))
```


Export formats

Exports can export to a given format:

```
class Format(models.Model):
    name = models.CharField(max_length=50)
    file_ext = models.CharField(max_length=10, blank=True)
    mime = models.CharField(max_length=50)
    template = models.CharField(max_length=255)
```

The mime field is the Content-Type needed for the response. file_ext will be used to compute the export's filename, provided via Content-Disposition header.

Example: let's take a naive export to csv:

- mime: text/csv
- file_ext: csv
- name: Naive CSV format
- template: data_exports/export_detail_csv.html (included as an example)

If an export uses this format, visiting the export's view page /exports/<export slug> will offer a file download, named <export slug>.csv.

Using your own views

To use your own views, you need to use the same url names as in data_exports/urls.py, and make sure they use the data_exports namespace, as django.core.urlresolvers.reverse is used internally to compute the needed urls.

You can check the included example views in data_exports/views.py, and of course reuse the forms provided in data_exports/forms.py.

Using your own templates

Django-data-exports makes use of Django’s template overloading mechanism. This means that if you provide a `data_exports/export_detail.html` template which has precedence over the one bundled with the app, it’ll be used.

Example: say you have a `templates/` folder in your project, and the appropriate `TEMPLATE_DIRS` setting. Place your own template in `project/templates/data_exports/export_detail.html` to have it used instead of the template bundled with the app in `data_exports/templates/data_exports/export_detail.html`.

There’s three included templates:

- `data_exports/base.html`: extended by the two other templates
- `data_exports/export_detail.html`: used by default for exports that don’t specify a format
- `data_exports/export_detail_csv.html`: used by the “naive csv format” detailed in [Export formats](#).

CHAPTER 5

Changes

- 0.2: fields on related models also available to exports
- 0.1: initial version

CHAPTER 6

Indices and tables

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