django CMS Export Objects Documentation

Release 0.1.0

Iacopo Spalletti

January 06, 2017

Contents

Contents:

CHAPTER 1

django CMS Export Objects

A django CMS command to export CMS Pages and PlaceholderFields-enabled objects and all their dependencies.

- Free software: BSD license

Warning: This is highly experimental, it's not guaranteed to work in any way, to keep your data intact; it may harm you, your cat and even create a black hole in a server near you. You've been warned!

1.1 Features

Largely based on django `dump_data` command it differs in two aspects:

- Objects selection is based on a queryset passed by the user (as string)
- Objects dependencies are evaluated on a per-instance base to dump only the corresponding dependent instances.

1.2 Documentation

For documentation see http://djangocms-export-objects.readthedocs.org

CHAPTER 2

Installation

Install using pip

```
$ pip install djangocms_export_objects
```

or directly from github:

```
$ pip install -e https://github.com/nephila/djangocms-export-objects#egg=djangocms_export_objects
```

CHAPTER 3

Usage

The main feature of *cms_dump_objects* is that you can select data to be exported using a normal Django QuerySet.

QuerySet must be written in quotes and can only contain scalar value as arguments to *filter*, *exclude* and other selection methods. Only methods that return a QuerySet can be used.

It share most other arguments with *dump_data*.

3.1 Examples

To dump the whole page tree with all the respective plugins and content:

```
./manage.py cms_dump_objects 'cms.Page.objects.all()' > /path/to/dump.json
```

To dump a branch starting for a given page:

```
./manage.py cms_dump_objects -r 'cms.Page.objects.filter(reverse_id=whatever)' > /path/to/dump.json
```

3.2 Options

- `--format`: Specifies the output serialization format for fixtures; by using Django serializer, you can use any serialization format enabled in your project.
- `--indent`: Specifies the indent level to use when pretty-printing output.
- `-n`, `--natural`: Use natural keys if they are available.
- `-a`, `--all`: Use Django's base manager to dump all models stored in the database, including those that would otherwise be filtered or modified by a custom manager. When dumping `Page` objects, `PageManager` is always used.

3.2.1 Options valid only when dumping Page objects

- `-r`, `--recursive`: it will fetch all the pages whose ancestor is in the given queryset (i.e.: it dumps branches of the page tree).
- `-s`, `--skip-ancestors`: Skip the ancestors of the pages in the given queryset. **Use this at your own risk**: You'll not be able do load back the data if you don't have **all** the ancestors of the dumped pages in the project you're loading this fixtures to.

3.3 Caveats

- As Django `loaddata` command matches objects based on their primary key, this tool is mostly intended as a way to do partial backup of existing projects, and to move data between different instances of the same project (say: development and testing environment and so on); using it to move data between projects can lead to data overwrite and data loss.
- When exporting a partial subset of pages, all the ancestors will be dumped too, to be able to load them back in a project; existing pages with the same primary keys in the target project will overwritten.
- To avoid the above behavior use `-s` option, but to be able to load data back you'll need to have the pages with the needed primary keys before loading.

CHAPTER 4

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:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/nephila/djangocms_export_objects/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.

4.1.2 Fix Bugs

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

4.1.3 Implement Features

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

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/nephila/djangocms_export_objects/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 :)

4.2 Get Started!

Ready to contribute? Here's how to set up *djangocms_export_objects* for local development.

1. Fork the *djangocms_export_objects* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/djangocms_export_objects.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 djangocms_export_objects
$ cd djangocms_export_objects/
$ python setup.py develop
```

4. Create a branch for local development:

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

Now you can make your changes locally.

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

```
  $ flake8 djangocms_export_objects tests
      $ python setup.py test
  $ tox

To get flake8 and tox, just pip install them into your virtualenv.
```

6. 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.

4.3 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 2.6, 2.7. Check https://travis-ci.org/nephila/djangocms_export_objects/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python runtests.py TestDjangocmsExportObjects
```

CHAPTER 5

Credits

5.1 Development Lead

- Iacopo Spalletti <i.spalletti@nephila.it>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.1.0

- Unreleased version

CHAPTER 7

Indices and tables

- genindex
- modindex
- search