drf_compound_fields Documentation

Release 0.2.0

Steven Cummings

December 07, 2016

Contents

Contents:

CHAPTER 1

Overview

Django-REST-framework serializer fields for compound types. Django-REST-framework provides the ability to deal with multiple objects using the *many=True* option on serializers. That allows for lists of objects and for fields to be lists of objects.

This package expands on that and provides fields allowing:

- Lists of simple (non-object) types, described by other serializer fields.
- Fields that allow values to be a list or individual item of some type.
- Dictionaries of simple and object types.
- Partial dictionaries which include keys specified in a list.

A quick example:

```
from drf_compound_fields.fields import DictField
from drf_compound_fields.fields import ListField
from drf_compound_fields.fields import ListOrItemField
from drf_compound_fields.fields import ListField
from rest_framework import serializers

class EmailContact(serializers.Serializer):
    email = serializers.EmailField()
    verified = serializers.BooleanField()

class UserProfile(serializers.Serializer):
    username = serializers.CharField()
    email_contacts = EmailContact(many=True)  # List of objects: possible with REST-framework alone
    # This is the new stuff:
    skills = ListField(serializers.CharField())  # E.g., ["javascript", "python", "ruby"]
    name = ListOrItemField(serializers.CharField())  # E.g., "Prince" or ["John", "Smith"]
    bookmarks = DictField(serializers.URLField())  # E.g., {"./": "http://slashdot.org"}
    measurements = PartialDictField(included_keys=['height', 'weight'], serializers.IntegerField())
```

See the usage for more information.

CHAPTER 2

Project info

- Free software: BSD license
- Documentation
- Source code
- Issue tracker
- CI server
- IRC: no channel but see AUTHORS for individual nicks on freenode.
- Mailing list: None yet, but please log an issue if you want to have discussions about this package.

CHAPTER 3

Installation

At the command line:

```
$ easy_install drf-compound-fields
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv drf-compound-fields
$ pip install drf-compound-fields
```

CHAPTER 4

Usage

The following sections explain how and when you would use each of the provided fields. Note that while the examples have been simplified to dictionary data, object conversions (via *restore_object* methods) are valid as well.

4.1 *ListField*

Signature:

```
ListField(item_field=None)
```

Use this field to create lists of simple types. If the item field is not given, then values are passed through as-is.

Declare a serializer with a list field:

```
from drf_compound_fields.fields import ListField
from rest_framework import serializers

class SkillsProfileSerializer(serializers.Serializer):
    name = serializers.CharField()
    skills = ListField(serializers.CharField(min_length=3))
```

Serialize an object with a list:

```
serializer = SkillsProfileSerializer({'name': 'John Smith', 'skills': ['Python', 'Ruby']})
print serializer.data
```

Output:

```
{'name': u'John Smith', 'skills': [u'Python', u'Ruby']}
```

Deserialize an object with a list:

```
serializer = SkillsProfileSerializer(data={'name': 'John Smith', 'skills': ['Python', 'Ruby']})
assert serializer.is_valid(), serializer.errors
print serializer.object
```

Output:

```
{'skills': ['Python', 'Ruby'], 'name': 'John Smith'}
```

Get validation errors from invalid data:

```
serializer = SkillsProfileSerializer(data={'name': 'John Smith', 'skills': ['Python', 'io']})
assert serializer.is_valid(), serializer.errors
```

Output:

```
Traceback (most recent call last):
  File "demo.py", line 36, in <module>
    assert serializer.is_valid(), serializer.errors
AssertionError: {'skills': [{1: [u'Ensure this value has at least 3 characters (it has 2).']}]}
```

NOTE You can technically pass serializers to *ListField*. However, since you can just tell a serializer to deal with multiple objects, it is recommended that you stick with this method.

4.2 ListOrItemField

Signature:

```
ListOrItemField(item_field=None)
```

A field whose values are either a value or lists of values described by the given item field. If the item field is not given, then values are passed through as-is.

Declare a serializer with a list-or-item field:

```
from drf_compound_fields.fields import ListField
from drf_compound_fields.fields import ListOrItemField
from rest_framework import serializers

class SkillsProfileSerializer(serializers.Serializer):
    name = serializers.CharField()
    skills = ListField(serializers.CharField(min_length=3))
    social_links = ListOrItemField(serializers.URLField())
```

Serialize with an item value:

```
print SkillsProfileSerializer({
    'name': 'John Smith',
    'skills': ['Python'],
    'social_links': 'http://chrp.com/johnsmith'
}).data
```

Output:

```
{'name': u'John Smith', 'skills': [u'Python'], 'social_links': u'http://chrp.com/johnsmith'}
```

Serialize with a list value:

```
data = SkillsProfileSerializer({
    'name': 'John Smith',
    'skills': ['Python'],
    'social_links': ['http://chrp.com/johnsmith', 'http://myface.com/johnsmith']
}).data
import pprint
pprint.pprint(data)
```

Output:

```
{'name': u'John Smith',
 'skills': [u'Python'],
 'social_links': [u'http://chrp.com/johnsmith', u'http://myface.com/johnsmith']}
```

Get validation errors for an item value:

```
serializer = SkillsProfileSerializer(data={
    'name': 'John Smith',
    'skills': ['Python'], 'social_links': 'not_a_url'
})
assert serializer.is_valid(), serializer.errors
```

Output:

```
Traceback (most recent call last):
  File "demo.py", line 23, in <module>
    assert serializer.is_valid(), serializer.errors
AssertionError: {'social_links': [u'Invalid value.']}
```

Get validation errors for a list value:

```
serializer = SkillsProfileSerializer(data={
    'name': 'John Smith',
    'skills': ['Python'],
    'social_links': ['http://chrp.com/johnsmith', 'not_a_url']
})
assert serializer.is_valid(), serializer.errors
```

Output:

```
Traceback (most recent call last):
  File "demo.py", line 23, in <module>
    assert serializer.is_valid(), serializer.errors
AssertionError: {'social_links': [{1: [u'Invalid value.']}]}
```

4.3 *DictField*

Signature:

```
DictField(child=None, unicode_options=None)
```

A field whose values are dicts of values described by the given value field. The value field can be another field type (e.g., CharField) or a serializer.

If *child* is not given, then the *dict* values are passed through-as-is, and can be anything. Note that in this case, any non-native *dict* values wouldn't be properly prepared for data rendering.

If given, unicode_options must be a dict providing options per the unicode function.

Dictionary keys are presumed to be character strings or convertible to such, and so during processing are casted to *unicode*. If necessary, options for unicode conversion (such as the encoding, or error processing) can be provided to a *DictField*. For more info, see the Python Unicode HOWTO.

Declare a serializer with a dict field:

```
from drf_compound_fields.fields import DictField
from rest_framework import serializers

class UserBookmarksSerializer(serializers.Serializer):
    username = serializer.CharField()
    links = DictField(serializers.URLField())
```

Serialize an object with a dict:

```
serializer = UserBookmarksSerializer({
    'username': 'jsmith',
    'links': {
        'Order of the Stick': 'http://www.giantitp.com/comics/oots.html',
        'The Hypertext Application Language': 'http://stateless.co/hal_specification.html'
    }
})
import pprint
pprint.pprint(serializer.data)
```

Output:

```
{'links': {u'Order of the Stick': u'http://www.giantitp.com/comics/oots.html',
           u'The Hypertext Application Language': u'http://stateless.co/hal_specification.html'},
 'username': u'jsmith'}
```

Deserialize an object with a dict:

```
serializer = UserBookmarksSerializer(data={
    'username': u'jsmith',
    'links': {
        'Order of the Stick': u'http://www.giantitp.com/comics/oots.html',
        'The Hypertext Application Language': u'http://stateless.co/hal_specification.html'
    }
})
assert serializer.is_valid(), serializer.errors
import pprint
pprint.pprint(serializer.object)
```

Output:

```
{'links': {u'Order of the Stick': u'http://www.giantitp.com/comics/oots.html',
           u'The Hypertext Application Language': u'http://stateless.co/hal_specification.html'},
 'username': u'jsmith'}
```

Get validation errors from invalid data:

```
serializer = UserBookmarksSerializer(data={
    'username': u'jsmith',
    'links': {
        'Order of the Stick': u'not_a_url',
        'The Hypertext Application Language': u'http://stateless.co/hal_specification.html'
    }
})
assert serializer.is_valid(), serializer.errors
```

Output:

```
Traceback (most recent call last):
  File "demo.py", line 25, in <module>
    assert serializer.is_valid(), serializer.errors
AssertionError: {'links': [{u'Order of the Stick': [u'Invalid value.']}]}
```

4.4 PartialDictField

Signature:

```
PartialDictField(included_keys, child=None, unicode_options=None)
```

A dict field whose values are filtered to only include values for the specified keys.

Declare a serializer with a partial dict field:

```
from drf_compound_fields.fields import PartialDictField
from rest_framework import serializers

class UserSerializer(serializers.Serializer):
    user_details = PartialDictField(['favorite_food'], serializers.CharField())
```

Serialize an object with a partial dict:

```
serializer = UserSerializer({'user_details': {'favorite_food': 'pizza', 'height': '52in'}})
import pprint
pprint.pprint(serializer.data)
```

Output:

```
{'user_details': {u'favorite_food': u'pizza'}}
```

Deserialize data with a partial dict:

```
serializer = UserSerializer(data{'user_details': {'favorite_food': 'pizza', 'height': '52in'}})
import pprint
pprint.pprint(serializer.object)
```

Output:

```
{'user_details': {u'favorite_food': u'pizza'}}
```

CHAPTER 5

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:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/estebistec/drf-compound-fields/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.

5.1.2 Fix Bugs

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

5.1.3 Implement Features

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

5.1.4 Write Documentation

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

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/estebistec/drf-compound-fields/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 :)

5.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/drf_compound_fields.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 drf_compound_fields
$ cd drf_compound_fields/
$ 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 drf_compound_fields 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.

5.3 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, and 3.3, and for PyPy. Check https://travis-ci.org/estebistec/drf_compound_fields/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_drf_compound_fields
```

CHAPTER 6

Credits

6.1 Development Lead

- Steven Cummings <cummingscs@gmail.com> (estebistec on freenode)

6.2 Contributors

- Sonny Hu <nullspace.hu@gmail.com>
- Victor Perron <public@iso3103.net>

CHAPTER 7

History

7.1 1.0.0 (2016-02-29)

- Upgrade ListOrItemField and PartialDictField for django-rest-framework 3.0 and beyond
- Remove ListField and DictField, as they now come with django-rest-framework

7.2 0.2.2 (2014-08-10)

Correct validation behaviors when fields are used in embedded serializers. Also correction to the *list* and *dict* type checks for *None* values (#15, #16, #18).

- Implement *initialize* and *field_from_native* to ensure proper validation in embedded serializers.
- Give the fields distinct *validate* and *run_validators* implementations that don't call each other.
- Don't apply the *list* and *dict* type checks for *None* values.

7.3 0.2.1 (2014-04-23)

Loosen dependency versions

- Remove explicit dependency on Django
- Loosen rest-framework to any version before 3

7.4 0.2.0 (2014-03-16)

- Documentation (#3)
- Collect messages of nested errors, instead of error objects (#12)
- Add ListOrItemField type (#5, #11)
- Fix PartialDictField validation and handling of badly-typed values
- Switch project tests to py.test (#10)

7.5 0.1.0 (2014-03-06)

First PyPI release of rest-framework serializer compound-fields (#1). Provides:

- ListField (#4, #7)
- DictField
- PartialDictField (#8, #9)

CHAPTER 8

Indices and tables

- genindex
- modindex
- search