Booby Documentation

Release 0.5.0

Jaime Gil de Sagredo

January 04, 2014

Contents

Booby is a standalone data *modeling* and *validation* library written in Python. Booby is under active development and licensed under the Apache2 license, so feel free to contribute and report errors and suggestions.

See the sample code below to get an idea of the main features.

```
from booby import Model, fields

class Token(Model):
    key = fields.String()
    secret = fields.String()

class User(Model):
    login = fields.String(required=True)
    name = fields.String()
    email = fields.Email()
    token = fields.Embedded(Token, required=True)
    addresses = fields.Field(default=list)

class Address(Model):
    line_1 = fields.String()
    line_2 = fields.String()

jack = User(
    login='jack',
    name='Jack',
    email='jack@example.com',
    token={
        'key': 'vs7dfxxx',
        'secret': 'ds5ds4xxx'
    },
    addresses=[
        Address(line_1='Main Street'),
        Address(line_1='Main St')
    ]
)

if jack.is_valid:
    print jack.to_json(indent=2)
else:
    print json.dumps(dict(jack.validation_errors))
```

```
{
  "email": "jack@example.com",
  "login": "jack",
  "token": {
    "secret": "ds5ds4xxx",
    "key": "vs7dfxxx"
  },
  "name": "Jack",
  "addresses": [
    {
      "line_1": "Main St",
      "line_2": null
    },
    {
      "line_1": "Main Street",
      "line_2": null
    }
  ]
```

```
}
```

CHAPTER 1

Installation

You can install the last stable release of Booby from PyPI using pip or easy_install.

```
$ pip install booby
```

Also you can install the latest sources from Github.

```
$ pip install -e git+git://github.com/jaimegildesagredo/booby.git#egg=booby
```

CHAPTER 2

Tests

To run the Booby test suite you should install the development requirements and then run nosetests.

```
$ pip install -r requirements-devel.txt
$ nosetests tests/unit
$ nosetests tests/integration
```

CHAPTER 3

Changes

See Changes.

CHAPTER 4

Documentation

Booby docs are hosted on Read The Docs.

CHAPTER 5

Contents

5.1 Installation

You can install Booby directly from PyPI using pip or easy_install:

```
$ pip install booby
```

Or install the latest sources from Github:

```
$ pip install -e git+git://github.com/jaimegildesagredo/booby.git#egg=booby
```

Also you can download a source code package from Github and install it using setuptools:

```
$ tar xvf booby-{version}.tar.gz
$ cd booby
$ python setup.py install
```

5.2 Models

The *models* module contains the *booby* highest level abstraction: the *Model*.

To define a model you should subclass the `Model` class and add a list of `fields` as attributes. And then you could instantiate your *Model* and work with these objects.

Something like this:

```
class Repo(Model):
     name = fields.String()
     owner = fields.Embedded(User)

booby = Repo(
    name='Booby',
    owner={
        'login': 'jaimegildesagredo',
        'name': 'Jaime Gil de Sagredo'
    })

print booby.to_json()
'{"owner": {"login": "jaimegildesagredo", "name": "Jaime Gil de Sagredo"}, "name": "Booby"}'
```

class models.**Model**(***kwargs*)

The *Model* class. All Booby models should subclass this.

By default the *Model's* `__init__()` takes a list of keyword arguments to initialize the *fields* values. If any of these keys is not a *field* then raises `errors.FieldError`. Of course you can overwrite the *Model's* `__init__()` to get a custom behavior.

You can get or set Model *fields* values in two different ways: through object attributes or dict-like items:

```
>>> booby.name is booby['name']
True
>>> booby['name'] = 'booby'
>>> booby['foo'] = 'bar'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
errors.FieldError: foo
```

Parameters ****kwargs** – Keyword arguments with the fields values to initialize the model.

update(**args*, ***kwargs*)

This method updates the *model* fields values with the given *dict*. The model can be updated passing a dict object or keyword arguments, like the Python's builtin `dict.update()`.

is_valid

This property will be *True* if there are not validation errors in this *model* fields. If there are any error then will be *False*.

This property wraps the `Model.validate()` method to be used in a boolean context.

validate()

This method validates the entire *model*. That is, validates all the `fields` within this model.

If some *field* validation fails, then this method raises the same exception that the `field.validate()` method had raised.

validation_errors

Generator of field name and validation error string pairs for each validation error on this *model* fields.

to_json(**args*, ***kwargs*)

This method returns the *model* as a *json string*. It receives the same arguments as the builtin `json.dump()` function.

To build a json representation of this *model* this method iterates over the object to build a *dict* and then serializes it as json.

5.3 Fields

The *fields* module contains a list of *Field* classes for model's definition.

The example below shows the most common fields and builtin validations:

```
class Token(Model):
    key = String()
    secret = String()

class User(Model):
    login = String(required=True)
```

```
    name = String()
    role = String(choices=['admin', 'moderator', 'user'])
    email = Email(required=True)
    token = Embedded(Token, required=True)
    is_active = Boolean(default=False)
```

`fields.`**`Field`**

This is the base class for all `booby.fields`. This class can also be used as field in any `models.Model` declaration.

Parameters

- **default** – This field *default*'s value.

  If passed a callable object then uses its return value as the field's default. This is particularly useful when working with mutable objects.

  If *default* is a callable it can optionaly receive the owner *model* instance as its first positional argument.
- **required** – If *True* this field value should not be *None*.
- **choices** – A *list* of values where this field value should be in.
- ***validators** – A list of field `validators` as positional arguments.

`fields.`**`String`**

`Field` subclass with builtin *string* validation.

`fields.`**`Integer`**

`Field` subclass with builtin *integer* validation.

`fields.`**`Float`**

`Field` subclass with builtin *float* validation.

`fields.`**`Boolean`**

`Field` subclass with builtin *bool* validation.

`fields.`**`Embedded`**

`Field` subclass with builtin embedded `models.Model` validation.

`fields.`**`Email`**

`Field` subclass with builtin *email* validation.

5.4 Validators

The *validators* module contains a set of `fields` validators.

A validator is any callable *object* which receives a *value* as the target for the validation. If the validation fails then should raise an `errors.ValidationError` exception with an error message.

Validators are passed to `fields.Field` and subclasses as possitional arguments.

`validators.`**`nullable`**(*method*)

This is a helper validation decorator for validators that allow their *values* to be `None`.

The `String` validator is a good example:

```
class String(object):
    def validate(self, value):
```

```
        if value is not None:
            pass # Do the validation here ...
```

Now the same but using the *@nullable* decorator:

```
@nullable
def validate(self, value):
    pass # Do the validation here ...
```

class validators.**Required**
This validator forces fields to have a value other than None.

class validators.**In**(*choices*)
This validator forces fields to have their value in the given list.

Parameters **choices** – A *list* of possible values.

class validators.**String**
This validator forces fields values to be an instance of *basestring*.

class validators.**Integer**
This validator forces fields values to be an instance of *int*.

class validators.**Float**
This validator forces fields values to be an instance of *float*.

class validators.**Boolean**
This validator forces fields values to be an instance of *bool*.

class validators.**Model**(*model*)
This validator forces fields values to be an instance of the given models.Model subclass and also performs a validation in the entire *model* object.

Parameters **model** – A subclass of models.Model

class validators.**Email**
This validator forces fields values to be strings and match a valid email address.

class validators.**List**(**validators*)
This validator forces field values to be a list. Also a list of inner validators could be specified to validate each list element. For example, to validate a list of models.Model you could do:

```
books = fields.Field(validators.List(validators.Model(YourBookModel)))
```

Parameters ***validators** – A list of inner validators as possitional arguments.

5.5 Inspection

The inspection module provides users and 3rd-party library developers a public api to access booby objects and classes internal data, such as defined fields, and some low-level type validations.

This module is based on the Python inspect module.

inspection.**get_fields**(*model*)
Returns a *dict* mapping the given *model* field names to their *fields.Field* objects.

Parameters **model** – The *models.Model* subclass or instance you want to get their fields.

Raises TypeError if the given *model* is not a model.

inspection.**is_model**(*obj*)
Returns *True* if the given object is a *models.Model* instance or subclass. If not then returns *False*.

5.6 Errors

The *errors* module contains all exceptions used by Booby.

exception errors.**BoobyError**
Base class for all Booby exceptions.

exception errors.**FieldError**
This exception is used as an equivalent to AttributeError for fields.

exception errors.**ValidationError**
This exception should be raised when a *value* doesn't validate. See validators.

5.7 Changes

5.7.1 0.5.0 (Jan 4, 2014)

Backwards-incompatible

- Now field *validators* must be callable objects. Before this release validators had a *validate* method that is not longer used to perform a validation. This change only affects to custom user validators with a *validate* method.

Highlights

- The *FieldError* exception now is raised only with the field name as argument. See issue 12.
- Fields *default* argument callables can now optionally receive the model as argument.
- Added the *inspection* module which provides the *get_fields* and *is_model* functions as a public api to get access to *models* fields and type validation.

5.7.2 0.4.0 (Ago 4, 2013)

Backwards-incompatible

- Moved the *Model.to_dict* functionality to *dict(model)*.
- The *Model.validation_errors* method now is an interable of field name and validaton error pairs.
- Removed the *Field* subfix for all Booby fields. Now use the module as namespace: *fields.String*.

Highlights

- Added an *is_valid* property to *Model*.
- The *Model* instances now are iterables of field name, value pairs.

5.7.3 0.3.0 (Jun 20, 2013)

Highlights

- When passed a *callable* object as a field *default* then the default value for this field in a model instance will be the return value of the given callable.
- Added the `models.Model.validation_errors()` method to get a dict of field name and error message pairs for all invalid model fields.

CHAPTER 6

Indices and tables

- *genindex*
- *modindex*
- *search*

Python Module Index