

---

# **Booby Documentation**

*Release 0.7.0*

**Jaime Gil de Sagredo**

October 07, 2015



<b>1 Usage</b>	<b>3</b>
<b>2 Installation</b>	<b>5</b>
<b>3 Tests</b>	<b>7</b>
<b>4 Changes</b>	<b>9</b>
<b>5 Contents</b>	<b>11</b>
5.1 Installation . . . . .	11
5.2 Models . . . . .	11
5.3 Fields . . . . .	12
5.4 Validators . . . . .	14
5.5 Inspection . . . . .	15
5.6 Errors . . . . .	15
5.7 Changes . . . . .	15
<b>6 Indices and tables</b>	<b>19</b>
<b>Python Module Index</b>	<b>21</b>



Booby is a standalone data *modeling* and *validation* library written in Python. Booby is under active development (visit [this blog post](#) for more info and the roadmap) and licensed under the [Apache2 license](#), so feel free to [contribute](#) and [report errors and suggestions](#).



---

## Usage

---

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

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

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



---

## 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
```



---

## Tests

---

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

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



---

**Changes**

---

See Changes.



## 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.

#### **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.

#### **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.

#### **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()`.

#### **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, but with the field name prepended.

#### **validation\_errors**

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

## 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)
```



```
class fields.Boolean(*args, **kwargs)
    Field subclass with builtin bool validation.
```

```
class fields.Collection(model, *args, **kwargs)
    Field subclass with builtin list of models.Model validation, encoding and decoding.
```

Example:

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

class User(Model):
    tokens = Collection(Token)

user = User({
    'tokens': [
        {
            'key': 'xxx',
            'secret': 'yyy'
        },
        {
            'key': 'zzz',
            'secret': 'xxx'
        },
    ]
})

user.tokens.append(Token(key='yyy', secret='xxx'))
```

```
class fields.Email(*args, **kwargs)
    Field subclass with builtin email validation.
```

```
class fields.Embedded(model, *args, **kwargs)
    Field subclass with builtin embedded models.Model validation.
```

```
class fields.Field(*validators, **kwargs)
    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 optionally 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.
- **name** – Specify an alternate key name to use when decoding and encoding.
- **read\_only** – If *True*, the value is treated normally in decoding but omitted during encoding.
- **\*validators** – A list of field *validators* as positional arguments.

```
class fields.Float (*args, **kwargs)
    Field subclass with builtin float validation.

class fields.Integer (*args, **kwargs)
    Field subclass with builtin integer validation.

class fields.List (*args, **kwargs)
    Field subclass with builtin list validation and default value.

class fields.String (*args, **kwargs)
    Field subclass with builtin string 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 positional arguments.

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

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

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

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

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 positional arguments.

```
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.Required
    This validator forces fields to have a value other than None.
```

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

## 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 0.7.0 (Dec 3, 2014)

#### Backwards-incompatible

- The `List` encoder no longer encodes models. To achieve the old behavior pass the `Model` encoder as an argument instead:

```
class User(Model):
    tokens = fields.Field(encoders=[encoders.List(encoders.Model())])
```

#### Highlights

- Added a `Collection` field that works like `Embedded` for lists of models:

```
class User(Model):
    tokens = fields.Collection(Token)

user = User({
    'tokens': [
        {
            'key': 'xxx',
            'secret': 'yyy'
```

```
        }
    ]
})

user.tokens.append(Token(key='zzz', secret='www'))
user.validate()
```

See [the docs](#) for more info.

### 5.7.2 0.6.0 (Oct 12, 2014)

#### Backwards-incompatible

- The *List* validator now accepts *None* as a valid value allowing not required list fields. Before this a field with a *List* validator couldn't be *None*.

#### Highlights

- The *Model* class now defines a *decode* and *encode* methods with serialization/deserialization support.
- A *Field* now can receive lists of callable objects, *encoders* and *decoders*, to perform serialization/deserialization.
- Added a *List* field that can be used to create fields containing lists of objects (even models).
- Datetime validator, encoder, and decoder were added.

### 5.7.3 0.5.2 (Mar 22, 2014)

#### Highlights

- Added readable *Field* instances repr. See [issue 20](#).
- Added readable *Model* classes and instances repr.

### 5.7.4 0.5.1 (Jan 31, 2014)

#### Highlights

- The *Email* validator now only performs a basic sanity check instead of the more restrictive previous check. See [issue 17](#).
- The *List* validator now accepts any object that implements the *list* interface (`collections.MutableSequence`). See [issue 18](#).
- Any object implementing the *dict* interface (`collections.MutableMapping`) can be used as a value for an *Embedded* field. See [issue 18](#).
- When iterating a *Model* object all objects implementing the *list* interface are treated as lists. See [issue 18](#).

### 5.7.5 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 no 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.

### 5.7.6 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 iterable of field name and validation error pairs.
- Removed the *Field* suffix 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.

### 5.7.7 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.



---

## Indices and tables

---

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





**e**

errors, 15

**f**

fields, 12

**i**

inspection, 15

**m**

models, 11

**v**

validators, 14



**B**

BoobyError, 15  
Boolean (class in fields), 13  
Boolean (class in validators), 14

**C**

Collection (class in fields), 13

**E**

Email (class in fields), 13  
Email (class in validators), 14  
Embedded (class in fields), 13  
errors (module), 15

**F**

Field (class in fields), 13  
FieldError, 15  
fields (module), 12  
Float (class in fields), 13  
Float (class in validators), 14

**G**

get\_fields() (in module inspection), 15

**I**

In (class in validators), 14  
inspection (module), 15  
Integer (class in fields), 14  
Integer (class in validators), 14  
is\_model() (in module inspection), 15  
is\_valid (models.Model attribute), 12

**L**

List (class in fields), 14  
List (class in validators), 14

**M**

Model (class in models), 11  
Model (class in validators), 14

models (module), 11

**R**

Required (class in validators), 14

**S**

String (class in fields), 14  
String (class in validators), 14

**T**

to\_json() (models.Model method), 12

**U**

update() (models.Model method), 12

**V**

validate() (models.Model method), 12  
validation\_errors (models.Model attribute), 12  
ValidationError, 15  
validators (module), 14