Carpentry Documentation

Release 0.1a

Samuele Santi

May 07, 2014

Contents:

CHAPTER 1

Usage

Creating your models is pretty straight-forward: just inherit from `carpentry.base.BaseObject` and define some fields.

```
from carpentry import BaseObject, StringField, BoolField

class User(BaseObject):
    first_name = StringField(required=True)
    last_name = StringField(required=True)
    email = StringField(required=True)
    is_administrator = BoolField(default=False)
```

Then you can start using it right away:

```
>>> user = User({'first_name': 'John', 'last_name': 'Doe'})
>>> user.email = 'john.doe@example.com'
```

And now serialize it and get ready for (inserting in a database | sending via an API | ...):

```
>>> obj.serialize()
{
'first_name': 'John',
'last_name': 'Doe',
'email': 'john.doe@example.com',
'is_administrator': False,
}
```

Also, fields provide basic validation, so if you try, for example, to:

```
>>> obj.is_administrator == 'FooBar'
```

You'll get a `TypeError`.

CHAPTER 2

API documentation

2.1 Base objects

class carpentry.base.**BaseObject**(*values=None*)

Base for the other objects, dispatching get/set/deletes to BaseField instances, if available.

__delattr__(*key*)

Custom attribute handling. If the attribute is a field, call its .del() method. Otherwise, perform the action directly on the object.

__eq__(*other*)

__getattribute__(*key*)

Custom attribute handling. If the attribute is a field, return the value returned from its .get() method. Otherwise, return it directly.

__init__(*values=None*)

Parameters **values** – Initial values for the object. Must be a dict or dict-like.

__repr__()

Provide a nice representation to objects, including serialized values.

__setattr__(*key*, *value*)

Custom attribute handling. If the attribute is a field, pass the value to its .set() method. Otherwise, set it directly on the object.

classmethod from_dict(*data*)

Deprecated since version 0.1: use normal constructor __init__() instead

is_equivalent(*other*, *ignore_key=True*)

Equivalency check between objects. Will make sure that values in all the non-key fields match.

Parameters

- **other** – other object to compare
- **ignore_key** – if set to True (the default), it will ignore "key" fields during comparison

is_modified()

The object is modified if any of its fields reports itself as modified.

classmethod iter_fields()

Iterate over fields in this objects, yielding (name, field) pairs.

serialize()

Create a serializable representation of the object.

set_initial(*values*)
Set initial values for all fields

to_dict()
Deprecated since version 0.1: use `serialize()` instead

class `carpentry.base.`**BaseField**(***kwargs*)
Pseudo-descriptor, accepting field names along with instance, to allow better retrieving data for the instance itself.

> **Warning:** Beware that fields shouldn't carry state of their own, a part from the one used for generic field configuration, as they are shared between instances.

__init__(***kwargs*)

Parameters

- **default** – Default value (if not callable) or function returning default value (if callable).
- **is_key** – Boolean indicating whether this is a key field or not. Key fields are ignored when comparing using `is_equivalent()`
- **required** – Mark the field as required, for validation purposes.

__repr__()

default = None

delete(*instance*, *name*)
Delete the modified value for a field (logically restores the original one)

get(*instance*, *name*)
Get the value for the field from the main instace, by looking at the first found in:

•the updated value

•the initial value

•the default value

get_default()

is_equivalent(*instance*, *name*, *other*, *ignore_key=True*)

is_key = False

is_modified(*instance*, *name*)
Check whether this field has been modified on the main instance.

serialize(*instance*, *name*)
Returns the "serialized" (json-encodable) version of the object.

set(*instance*, *name*, *value*)
Set the modified value for a field

set_initial(*instance*, *name*, *value*)
Set the initial value for a field

validate(*instance*, *name*, *value*)
The validate method should be the (updated) value to be used as the field value, or raise an exception in case it is not acceptable at all.

2.2 Common fields

class carpentry.fields.**StringField**(***kwargs*)

default = ''

validate(*instance*, *name*, *value*)

class carpentry.fields.**BoolField**(***kwargs*)

default = False

validate(*instance*, *name*, *value*)

class carpentry.fields.**IntegerField**(***kwargs*)

default = 0

validate(*instance*, *name*, *value*)

class carpentry.fields.**ListField**(***kwargs*)

static **default**()

validate(*instance*, *name*, *value*)

class carpentry.fields.**SetField**(***kwargs*)

static **default**()

serialize(*instance*, *name*)

validate(*instance*, *name*, *value*)

class carpentry.fields.**DictField**(***kwargs*)

static **default**()

validate(*instance*, *name*, *value*)

CHAPTER 3

Indices and tables

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

C