marshmallow-polyfield Documentation

Release latest

September 13, 2015

An unofficial extension to Marshmallow to allow for polymorphic fields.

Marshmallow is a fantastic library for serialization and deserialization of data. For more on that project see its GitHub page or its Documentation.

This project adds a custom field designed for polymorphic types. This allows you to define a schema that says "This field accepts anything of type X"

The secret to this field is that you need to define two functions. One to be used when serializing, and another for deserializing. These functions take in the raw value and return the schema to use.

This field should support the same properties as other Marshmallow fields. I have worked with *required allow_none* and *many*.

CHAPTER 1

Installing

```
$ pip install marshmallow-polyfield
```

CHAPTER 2

Importing

Here is how to import the necessary field class

```
from marshmallow_polyfield import PolyField
```

CHAPTER 3

Example

The code below demonstrates how to setup a schema with a PolyField. For the full context check out the tests. Once setup the schema should act like any other schema. If it does not then please file an Issue.

```
def shape_schema_serialization_disambiguation(base_object):
    class_to_schema = {
        Rectangle.__name__: RectangleSchema,
        Triangle.__name__: TriangleSchema
    }
    try:
        return class_to_schema[base_object.__class__.__name__]()
    except KeyError:
        pass

    raise TypeError("Could not detect type. "
                    "Did not have a base or a length. "
                    "Are you sure this is a shape?")


def shape_schema_deserialization_disambiguation(object_dict):
    if object_dict.get("base"):
        return TriangleSchema()
    elif object_dict.get("length"):
        return RectangleSchema()

    raise TypeError("Could not detect type. "
                    "Did not have a base or a length. "
                    "Are you sure this is a shape?")

class ContrivedShapeClass(object):
    def __init__(self, main, others):
        self.main = main
        self.others = others

    def __eq__(self, other):
        return self.__dict__ == other.__dict__


class ContrivedShapeClassSchema(Schema):
    main = PolyField(
        serialization_schema_selector=shape_schema_serialization_disambiguation,
        deserialization_schema_selector=shape_schema_deserialization_disambiguation,
        required=True
    )
    others = PolyField(
        serialization_schema_selector=shape_schema_serialization_disambiguation,
        deserialization_schema_selector=shape_schema_deserialization_disambiguation,
        allow_none=True,
        many=True
    )
    def make_object(self, data):
        return TestPolyField.ContrivedShapeClass(
            data.get('main'),
            data.get('others')
        )
```