Just Objects

Release 0.3.0rc3.dev1

Rowland Ogwara

Dec 05, 2021

CONTENTS:

A simple python data objects management and validation tool based on jsonschema standards.

CHAPTER

ONE

REQUIREMENTS

- Python 3.6+

CHAPTER

TWO

OBJECTIVES

1. Define and demarcate data objects with just python annotations
2. Define constraints in simple jsonschema compliant manner
3. Validate data objects using standard jsonschema validators
4. Express complete jsonschema as simple data objects (its just objects)

CHAPTER

THREE

SIMILAR PROJECTS

- pydantic
- marshmallow

CHAPTER

FOUR

INSTALL

install from pip

```
$ pip install justobjects
```

CHAPTER

FIVE

USAGE EXAMPLE

```
import json
import justobjects as jo


@jo.data(typed=True)
class Model:
    a: int
    b: float
    c: str


# display schema
print(json.dumps(jo.show_schema(Model), indent=2))


try:
    # fails validation
    Model(a=3.1415, b=2.72, c="123")
except jo.schemas.ValidationException as err:
    print(err.errors)
```

5.1 Data Objects

A data object is a python class annotated with `@jo.data`. All data objects are automatically associated with a json schema based on the properties of the class. The associated json schema is used for validation.

5.1.1 Define data objects

```
import json
from typing import Iterable, List

import justobjects as jo


@jo.data()
class Model:
    a = jo.integer(minimum=3, maximum=30, multiple_of=3)
```

(continues on next page)

(continued from previous page)

```
    b = jo.numeric(default=0.3, multiple_of=2)
    c = jo.string(default="123")


# display schema
print(json.dumps(jo.show_schema(Model), indent=2))
```

This will output

```
{
  "type": "object",
  "title": "Draft7 JustObjects schema for data object '__main__.Model'",
  "additionalProperties": false,
  "properties": {
    "$schema": {
      "type": "string",
      "default": "http://json-schema.org/draft-07/schema#"
    },
    "a": {
      "type": "integer",
      "maximum": 30,
      "minimum": 3,
      "multipleOf": 3
    },
    "b": {
      "type": "number",
      "default": 0.3,
      "multipleOf": 2
    },
    "c": {
      "type": "string",
      "default": "123"
    }
  }
}
```

Validate Instances

Validation can be performed on model instances like this

```
try:
    # fails validation
    Model(a=3.1415, b=2.72, c="123")
except jo.ValidationException as err:
    print(err.errors)
```

validation can also be performed on dictionary instances too

```
try:
    # fails validation
    jo.validate(Model, dict(a=3.1415, b=2.72, c="123"))
except jo.ValidationException as err:
    print(err.errors)
```

5.1.2 Object Fields

Class fields can be defined using either of the following:

- PEP-526 annotations on Python 3.6+
- type arguments using `jo` field types

PEP 526 fields definitions

`typed=True` flag can be used to signify the model properties are defined using type annotations.

Note: Type annotations can only be used for basic constraint definitions. For example they cannot be used to defined custom constraints on string fields like maxLenght or pattern. For these the provided custom field types are more suitable.

```
@jo.data(typed=True)
class TypedModel:
    a: int
    b: float = 0.3
    c: str = "123"
```

5.2 Field Types

justobjects provides support for: - standard library types - custom justobjects type arguments - custom justonjects classes that can be used as typed annotations

5.2.1 Standard Library Types

Standard library types can be used to annotated properties in data objects

5.2.2 Types

justobjects provides multiple types that can be used to annotated fields

StringType

Generates a jsonschema with `type = "string"`

`justobjects.decorators.all_of`(*types: Iterable[Type]*, *default: Optional[Any] = None*, *required: bool = False*, *description: Optional[str] = None*) → attr._make.Attribute

JSON schema allOf

`justobjects.decorators.any_of`(*types: Iterable[Type]*, *default: Optional[Any] = None*, *required: bool = False*, *description: Optional[str] = None*) → attr._make.Attribute

JSON schema anyOf

`justobjects.decorators.array`(*item: Type*, *contains: bool = False*, *min_items: Optional[int] = 1*, *max_items: Optional[int] = None*, *required: bool = False*, *unique_items: bool = False*, *description: Optional[str] = None*) → attr._make.Attribute

Array schema data type

If *item* is the class type of another data object, it will be converted to a reference

Parameters

- **item** – data object class type used as items in the array
- **contains** – schema only needs to validate against one or more items in the array.
- **min_items** – positive integer representing the minimum number of items that can be on the array
- **max_items** – positive integer representing the maximum number of items that can be on the array
- **required** – True if field is required
- **unique_items** – disallow duplicates
- **description** – field description

Returns A array attribute wrapper

`justobjects.decorators.boolean`(*default: Optional[bool] = None*, *required: Optional[bool] = None*, *description: Optional[str] = None*) → attr._make.Attribute

Boolean schema data type

Parameters

- **default** – default boolean value
- **required** (*bool*) –
- **description** (*str*) – summary/description

Returns boolean schema wrapper

`justobjects.decorators.integer`(*default: Optional[int] = None*, *minimum: Optional[int] = None*, *maximum: Optional[int] = None*, *multiple_of: Optional[int] = None*, *exclusive_min: Optional[int] = None*, *exclusive_max: Optional[int] = None*, *required: Optional[bool] = None*, *description: Optional[str] = None*) → attr._make.Attribute

The integer type is used for integral numbers

Parameters

- **default** – default value used for instances
- **minimum** – a number denoting the minimum allowed value for instances
- **maximum** – a number denoting the maximum allowed value for instances
- **multiple_of** – must be a positive value, restricts values to be multiples of the given number
- **exclusive_max** – a number denoting maximum allowed value should be less that the given value
- **exclusive_min** – a number denoting minimum allowed value should be greater that the given value
- **required** – True if field should be a required field
- **description** – Comments describing the field

Returns A wrapped IntegerType

justobjects.decorators.**numeric**(*default: Optional[float] = None*, *minimum: Optional[float] = None*, *maximum: Optional[float] = None*, *multiple_of: Optional[int] = None*, *exclusive_min: Optional[float] = None*, *exclusive_max: Optional[float] = None*, *required: Optional[bool] = None*, *description: Optional[str] = None*) → attr._make.Attribute

The number type is used for any numeric type, either integers or floating point numbers.

Parameters

- **default** – default value used for instances
- **minimum** – a number denoting the minimum allowed value for instances
- **maximum** – a number denoting the maximum allowed value for instances
- **multiple_of** – must be a positive value, restricts values to be multiples of the given number
- **exclusive_max** – a number denoting maximum allowed value should be less that the given value
- **exclusive_min** – a number denoting minimum allowed value should be greater that the given value
- **required** – True if field should be a required field
- **description** – Comments describing the field

Returns A wrapped NumericType

justobjects.decorators.**one_of**(*types: Iterable[Type]*, *default: Optional[Any] = None*, *required: bool = False*, *description: Optional[str] = None*) → attr._make.Attribute

Applies to properties and complies with JSON schema oneOf property :param types: list of types that will be allowed :type types: list[type] :param default: default object instance that must be one of the allowed types :type default: object :param required: True if property is required :param description: field comments/description

Returns field instance

Return type attr.ib

justobjects.decorators.**ref**(*ref_type: Type*, *required: bool = False*, *description: Optional[str] = None*, *default: Optional[Type] = None*) → attr._make.Attribute

Creates a json reference to another json object

Parameters

- **ref_type** – class type referenced
- **required** – True if field is required
- **description** – ref specific documentation/comments
- **default** – default value

Returns a schema reference attribute wrapper

5.3 schemas

justobjects.schemas.**get_schema**(*cls: Union[Type[justobjects.types.JustSchema], justobjects.types.RefType, justobjects.types.BasicType]*) → Union[justobjects.types.JustSchema, justobjects.types.SchemaType]

Retrieves a justschema representation for the class or object instance

Parameters **cls** – a class type which is expected to be a pre-defined data object or an instance of json type

justobjects.schemas.**show_schema**(*model: Any*) → Dict

Converts a data object class type into a valid json schema

Parameters **model** – data object class type or instance

Returns a json schema dictionary

Examples

Creating and getting the schema associated with a simple integer type

```
import justobjects as jo
s = jo.IntegerType(minimum=3)
jo.show_schema(s)
# {'minimum': 3, 'type': 'integer'}
```

justobjects.schemas.**transform**(*cls: Type*) → justobjects.types.JustSchema

"Attempts to transform any object class type into an appropriate schema type

justobjects.schemas.**validate**(*schema: justobjects.types.JustSchema*, *instance: Any*) → None
justobjects.schemas.**validate**(*schema: Type*, *instance: Any*) → None
justobjects.schemas.**validate**(*schema: Any*, *instance: Any = None*) → None

Validates an object instance against its associated json schema

Parameters

- **schema** – a data object schema instance
- **instance** – data object instance

Raises **ValidationException** – when there errors

Examples

```
import justobjects as jo

@jo.data()
class Model:
  a = jo.integer(minimum=18)
  b = jo.boolean()

jo.validate(Model(a=4, b=True)
```

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

j

INDEX