JSL

Release 0.0.4

February 19, 2015

Contents

Release v0.0.4.

CHAPTER 1

Introduction

JSON Schema and its Python implementation are wonderful tools for data validation.

JSL is a Python library that provides a DSL for describing JSON schemas.

Why? Well, JSON schemas, especially large ones, can be tiresome to write. The standard is not always intuitive and leaves some places to make a mistake – for instance, it is easy to mix `maxItems` keyword with `maxLength`, or to forget to set `additionalProperties` to false, and so on. The syntax is not very concise and sometimes schema definitions get clumsy and hard to comprehend.

The DSL allows you to define a JSON schema in the way similar to how you define a model using an ORM – using classes and fields and relying on some metaclass magic under the hood.

It:

- makes reading and writing schemas easier;
- makes easier to decompose large schema definitions into smaller readable pieces;
- makes schemas extendable using the class inheritance;
- enables the autocomplete feature of your IDE;
- prevents you from making mistypes in JSON schema keywords (by throwing a RuntimeError);
- ...

Let's take a look at examples.

CHAPTER 2

Examples

```
from jsl import Document, StringField, ArrayField, DocumentField, OneOfField

class Entry(Document):
    name = StringField(required=True)

class File(Entry):
    content = StringField(required=True)

class Directory(Entry):
    content = ArrayField(OneOfField([
        DocumentField(File, as_ref=True),
        DocumentField('self')
    ]), required=True)
```

`Directory.to_schema()` will return the following schema:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "definitions": {
        "module.File": {
            "type": "object",
            "additionalProperties": false,
            "required": [
                "content",
                "name"
            ],
            "properties": {
                "content": {"type": "string"},
                "name": {"type": "string"}
            }
        },
        "module.Directory": {
            "type": "object",
            "additionalProperties": false,
            "required": [
                "content",
                "name"
            ],
            "properties": {
                "content": {
                    "type": "array",
                    "items": {
                        "oneOf": [
```

```
                            {"$ref": "#/definitions/module.File"},
                            {"$ref": "#/definitions/module.Directory"}
                        ]
                    }
                },
                "name": {"type": "string"}
            }
        }
    },
    "$ref": "#/definitions/module.Directory"
}
```

A JSON schema from the official documentation, defined using JSL:

```
from jsl import Document, BooleanField, StringField, ArrayField, DocumentField, OneOfField, IntField

class DiskDevice(Document):
    type = StringField(enum=['disk'], required=True)
    device = StringField(pattern='^/dev/[^/]+(/[^/]+)*$', required=True)

class DiskUUID(Document):
    type = StringField(enum=['disk'], required=True)
    label = StringField(pattern='^[a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F
                        required=True)

class NFS(Document):
    type = StringField(enum=['nfs'], required=True)
    remotePath = StringField(pattern='^(/[^/]+)+$', required=True)
    server = OneOfField([
        StringField(format='ipv4'),
        StringField(format='ipv6'),
        StringField(format='host-name'),
    ], required=True)

class TmpFS(Document):
    type = StringField(enum=['tmpfs'], required=True)
    sizeInMb = IntField(minimum=16, maximum=512, required=True)

class Schema(Document):
    class Options(object):
        schema_uri = 'http://json-schema.org/draft-04/schema#'
        description = 'schema for an fstab entry'

    storage = OneOfField([
        DocumentField(DiskDevice, as_ref=True),
        DocumentField(DiskUUID, as_ref=True),
        DocumentField(NFS, as_ref=True),
        DocumentField(TmpFS, as_ref=True),
    ], required=True)
    fstype = StringField(enum=['ext3', 'ext4', 'btrfs'])
    options = ArrayField(StringField(), min_items=1, unique_items=True)
    readonly = BooleanField()
```

CHAPTER 3

Installation

```
$ pip install jsl
```

CHAPTER 4

API

4.1 Document

class jsl.document.**Options**(*additional_properties=False*, *pattern_properties=None*, *min_properties=None*, *max_properties=None*, *title=None*, *description=None*, *default=None*, *enum=None*, *definition_id=None*, *schema_uri=u'http://json-schema.org/draft-04/schema#'*)

A container for options. Its primary purpose is to create an instance of options for every instance of a Document.

All the arguments are the same and work exactly as for fields.DictField except these:

Parameters

- **definition_id** (*str*) – A unique string to be used as a key for this document in the "definitions" schema section. If not specified, will be generated using module and class names.
- **schema_uri** (*str*) – An URI of the JSON Schema meta-schema.

class jsl.document.**Document**

A document. Can be thought as a kind of fields.DictField, which properties are defined by the fields added to the document class.

It can be tuned using special Options attribute (see Options for available settings).

Example:

```
class User(Document):
    class Options(object):
        title = 'User'
        description = 'A person who uses a computer or network service.'
    login = StringField(required=True)
```

classmethod get_schema()

Returns a JSON schema (draft v4) of the document.

4.2 Fields

4.2.1 Base Fields

class jsl.fields.**BaseField**(*required=False*)

A base class for fields in a JSL document.Document. Instances of this class may be added to a document to

define its properties.

Parameters **required** – If the field is required, defaults to False.

get_schema()
Returns a JSON schema (draft v4) of the data described by this field.

class jsl.fields.**BaseSchemaField**(*default=None*, *enum=None*, *title=None*, *description=None*, ***kwargs*)
A base class for fields that directly map to JSON Schema validator.

Parameters

- **required** – If the field is required, defaults to False.
- **default** – The default value for this field. May be a callable.
- **enum** – A list of valid choices. May be a callable.
- **title** – A short explanation about the purpose of the data described by this field.
- **description** – A detailed explanation about the purpose of the data described by this field.

4.2.2 Schema Fields

class jsl.fields.**DocumentField**(*document_cls*, *as_ref=False*, ***kwargs*)
A reference to a nested document.

Parameters

- **document_cls** – A string (dot-separated path to document class, i.e. 'app.resources.User'), `RECURSIVE_REFERENCE_CONSTANT` or a `Document`
- **as_ref** – If true, `document_cls`'s schema is placed into the definitions section, and the field schema is just a reference to it: `{"$ref": "#/definitions/..."}`. Makes a resulting schema more readable.

class jsl.fields.**ArrayField**(*items*, *min_items=None*, *max_items=None*, *unique_items=False*, *additional_items=None*, ***kwargs*)
An array field.

Parameters

- **items** – Either of the following:
  - `BaseField` – all items of the array must match the field schema;
  - a list or a tuple of `BaseField` s – all items of the array must be valid according to the field schema at the corresponding index (tuple typing).
- **min_items** (*int*) – A minimum length of an array.
- **max_items** (*int*) – A maximum length of an array.
- **unique_items** (bool or `BaseField`) – Whether all the values in the array must be distinct.
- **additional_items** – If the value of `items` is a list or a tuple, and the array length is larger than the number of fields in `items`, then the additional items are described by the schema in this property.

class jsl.fields.**DictField**(*properties=None*, *pattern_properties=None*, *additional_properties=None*, *min_properties=None*, *max_properties=None*, ***kwargs*)
A dictionary field.

Parameters

- **properties** (dict from str to `BaseField`) – A dictionary containing fields.
- **pattern_properties** (dict from str to `BaseField`) – A dictionary whose keys are regular expressions (ECMA 262). Properties match against these regular expressions, and for any that match, the property is described by the corresponding field schema.
- **additional_properties** (bool or `BaseField`) – Describes properties that are not described by the `properties` or `pattern_properties`.
- **min_properties** – A minimum number of properties.
- **max_properties** – A maximum number of properties

class `jsl.fields.`**`NotField`**(*field*, ***kwargs*)

Parameters **field** (`BaseField`) – a field to negate

class `jsl.fields.`**`OneOfField`**(*fields*, ***kwargs*)

Parameters **fields** (list of `BaseField`) – a list of fields, exactly one of which describes the data

class `jsl.fields.`**`AnyOfField`**(*fields*, ***kwargs*)

Parameters **fields** (list of `BaseField`) – a list of fields, at least one of which describes the data

class `jsl.fields.`**`AllOfField`**(*fields*, ***kwargs*)

Parameters **fields** (list of `BaseField`) – a list of fields, all of which describe the data

class `jsl.fields.`**`BooleanField`**(*default=None*, *enum=None*, *title=None*, *description=None*, ***kwargs*)

A boolean field.

class `jsl.fields.`**`StringField`**(*pattern=None*, *format=None*, *min_length=None*, *max_length=None*, ***kwargs*)

A string field.

Parameters

- **pattern** (*string*) – A regular expression (ECMA 262) that a string value must match.
- **format** (*string*) – A semantic format of the string (for example, "date-time", "email", or "uri").
- **min_length** (*int*) – A minimum length.
- **max_length** (*int*) – A maximum length.

class `jsl.fields.`**`EmailField`**(*pattern=None*, *format=None*, *min_length=None*, *max_length=None*, ***kwargs*)

An email field.

class `jsl.fields.`**`IPv4Type`**(*pattern=None*, *format=None*, *min_length=None*, *max_length=None*, ***kwargs*)

An IPv4 field.

class `jsl.fields.`**`DateTimeField`**(*pattern=None*, *format=None*, *min_length=None*, *max_length=None*, ***kwargs*)

An ISO 8601 formatted date-time field.

class `jsl.fields.`**`UriField`**(*pattern=None*, *format=None*, *min_length=None*, *max_length=None*, ***kwargs*)

A URI field.

class jsl.fields.**UriField**(*pattern=None*, *format=None*, *min_length=None*, *max_length=None*, ***kwargs*)

A URI field.

class jsl.fields.**NumberField**(*multiple_of=None*, *minimum=None*, *maximum=None*, *exclusive_minimum=False*, *exclusive_maximum=False*, ***kwargs*)

A number field.

Parameters

- **multiple_of** – A value must be a multiple of this factor.
- **minimum** – A minimum allowed value.
- **exclusive_minimum** – Whether a value is allowed to exactly equal the minimum.
- **maximum** – A maximum allowed value.
- **exclusive_maximum** – Whether a value is allowed to exactly equal the maximum.

class jsl.fields.**IntField**(*multiple_of=None*, *minimum=None*, *maximum=None*, *exclusive_minimum=False*, *exclusive_maximum=False*, ***kwargs*)

An integer field.

CHAPTER 5

Contributing

The project is hosted on GitHub. Please feel free to send a pull request or open an issue.

5.1 Running the Tests

```
$ pip install -r ./requirements-dev.txt
$ ./test.sh
```

A

B

D

E

G

I

N

O

S

U