Json Extensions Documentation

Release

Xavier Barbosa

July 07, 2015

Contents

This library implements tools inspired by several cleaver specifications around JSON.

Features:

jsonspec.cli Expose JSON Pointer , JSON Schema and JSON Patch to your console:

```
json extract '#/foo/1' --document-json='{"foo": ["bar", "baz"]}'
json validate --schema-file=schema.json < doc.json
cat doc.json | json add '#/foo/1' --fragment='{"foo": ["bar", "baz"]}'
```

jsonspec.pointer Implements JSON Pointer and Relative JSON Pointer, it offers a way to target a subelement.

jsonspec.reference Implements JSON Reference and offers a way to cross reference json documents.

jsonspec.operations Inspired by JSON Patch, it gives the ability to manipulate the document tree.

jsonspec.validators Implements JSON Schema, it adds the power of document validation.

Of course, it works for Python 2.7, Python 3.3 and Python 3.4.

```python
from jsonspec.validators import load

# data will validate against this schema
validator = load({
    'type': 'object',
    'properties': {
        'firstName': {
            'type': 'string',
        },
        'lastName': {
            'type': 'string',
        },
        'age': {
            'type': 'integer'
        }
    },
    'required': ['firstName', 'lastName', 'age']
})

# validate this data
validator.validate({
    'firstName': 'John',
    'lastName': 'Noone',
    'age': 33,
})
```

CHAPTER 1

Documentation

1.1 Installation

This package can be used by python 2.7, 3.3 and pypi.

Install from the Cheese shop:

```
$ pip install json-spec
```

Install from the sources:

```
$ pip install git+https://github.com/johnnoone/jsonspec.git
```

1.2 Command-line interface

All commands are suitable for bash scriptings :

- they return real error code.
- json documents can be feed with pipelines.

1.2.1 json add

Transform a json document.

Usage

```
json add [-h] [--document-json <doc> | --document-file <doc>]
         [--fragment-json <fragment> | --fragment-file <fragment>]
         [--indent <indentation>]
         <pointer>
```

Examples

```
json add '#/foo/1' --fragment-file=fragment.json --document-json='{"foo": ["bar", "baz"]}'
echo '{"foo": ["bar", "baz"]}' | json add '#/foo/1' --fragment-json='first'
json add '#/foo/1' --fragment-file=fragment.json --document-file=doc.json
json add '#/foo/1' --fragment-file=fragment.json < doc.json
```

1.2.2 json check

Tests that a value at the target location is equal to a specified value.

Usage

```
json check [-h] [--document-json <doc> | --document-file <doc>]
           [--fragment-json <fragment> | --fragment-file <fragment>]
           <pointer>
```

Examples

```
json check '#/foo/1' --fragment-file=fragment.json --document-json='{"foo": ["bar", "baz"]}'
echo '{"foo": ["bar", "baz"]}' | json check '#/foo/1' --fragment-file=fragment.json
json check '#/foo/1' --fragment-file=fragment.json --document-file=doc.json
json check '#/foo/1' --fragment-file=fragment.json < doc.json
```

1.2.3 json copy

Copies the value at a specified location to the target location.

Usage

```
json copy [-h] [--document-json <doc> | --document-file <doc>]
          [-t <target>] [--indent <indentation>]
          <pointer>
```

Examples

```
json copy '#/foo/1' --target='#/foo/2' --document-json='{"foo": ["bar", "baz"]}'
echo '{"foo": ["bar", "baz"]}' | json copy '#/foo/1' --target='#/foo/2'
json copy '#/foo/1' --target='#/foo/2' --document-file=doc.json
json copy '#/foo/1' --target='#/foo/2' < doc.json
```

1.2.4 json extract

Extract a fragment from a json document.

Usage

```
json extract [-h] [--document-json <doc> | --document-file <doc>]
             [--indent <indentation>]
             <pointer>
```

Examples

```
json extract '#/foo/1' --document-json='{"foo": ["bar", "baz"]}'
echo '{"foo": ["bar", "baz"]}' | json extract '#/foo/1'
json extract '#/foo/1' --document-file=doc.json
json extract '#/foo/1' < doc.json
```

1.2.5 json move

Removes the value at a specified location and adds it to the target location.

Usage

```
json move [-h] [--document-json <doc> | --document-file <doc>]
          [-t <target>] [--indent <indentation>]
          <pointer>
```

Examples

```
json move '#/foo/2' --target='#/foo/1' --document-json='{"foo": ["bar", "baz"]}'
echo '{"foo": ["bar", "baz"]}' | json move '#/foo/2' --target='#/foo/1'
json move '#/foo/2' --target='#/foo/1' --document-file=doc.json
json move '#/foo/2' --target='#/foo/1' < doc.json
```

1.2.6 json remove

Removes the value at a specified location and adds it to the target location.

Usage

```
json remove [-h] [--document-json <doc> | --document-file <doc>]
            [--indent <indentation>]
            <pointer>
```

Examples

```
json remove '#/foo/1' --document-json='{"foo": ["bar", "baz"]}'
echo '{"foo": ["bar", "baz"]}' | json remove '#/foo/1'
json remove '#/foo/1' --document-file=doc.json
json remove '#/foo/1' < doc.json
```

1.2.7 json replace

Removes the value at a specified location and adds it to the target location.

Usage

```
json replace [-h] [--document-json <doc> | --document-file <doc>]
             [--fragment-json <fragment> | --fragment-file <fragment>]
             [--indent <indentation>]
             <pointer>
```

Examples

```
json replace '#/foo/1' --fragment-file=fragment.json --document-json='{"foo": ["bar", "baz"]}'
echo '{"foo": ["bar", "baz"]}' | json replace '#/foo/1' --fragment-file=fragment.json
json replace '#/foo/1' --fragment-file=fragment.json --document-file=doc.json
json replace '#/foo/1' --fragment-file=fragment.json < doc.json
```

1.2.8 json validate

Validate document against a schema.

Usage

```
json validate [-h] [--document-json <doc> | --document-file <doc>]
              [--schema-json <schema> | --schema-file <schema>]
              [--indent <indentation>]
```

Examples

```
json validate --schema-file=schema.json --document-json='{"foo": ["bar", "baz"]}'
echo '{"foo": ["bar", "baz"]}' | json validate --schema-file=schema.json
json validate --schema-file=schema.json --document-file=doc.json
json validate --schema-file=schema.json < doc.json
```

1.3 JSON Pointer

JSON Pointer defines a string syntax for identifying a specific value within a JSON document. The most common usage is this:

```
from jsonspec.pointer import extract, Pointer
document = {
    'foo': ['bar', 'baz', {
        '$ref': 'obj2#/sub'
    }]
}
assert 'baz' == extract(document, '/foo/1')
```

But you can also iter throught the object:

```
obj = document
for token in Pointer('/foo/1'):
    obj = token.extract(obj)
assert 'baz' == obj
```

This module is event driven. It means that an event will be raised when it can't be explored. Here is the most meaningful:

Event	meaning
RefError	encountered a JSON Reference, *see*.
LastElement	asking for the last element of a sequence
OutOfBounds	member does not exists into the current mapping
OutOfRange	element does not exists into the current sequence

About JSON Reference

A *pointer.RefError* is raised when when a JSON Reference is encountered. This behavior can be desactivated by setting `bypass_ref=True`.

```
assert 'obj2#/sub' == extract(document, '/foo/2/$ref', bypass_ref=True)
```

If you need to resolve JSON Reference, you can that a look at JSON Reference.

About relative JSON Reference

Relative JSON Pointer are still experimental, but this library offers an implementation of it.

It implies to convert the whole document into a staged document, and then follow these rules:

```
from jsonspec.pointer import extract, stage

staged_doc = stage({
    'foo': ['bar', 'baz'],
    'highly': {
        'nested': {
            'objects': True
        }
    }
})

baz_relative = extract(self.document, '/foo/1')

# staged objects
assert extract(baz_relative, '0') == 'baz'
assert extract(baz_relative, '1/0') == 'bar'
assert extract(baz_relative, '2/highly/nested/objects') == True  # `foo is True` won't work

# keys, not staged
assert extract(baz_relative, '0#') == 1
assert extract(baz_relative, '1#') == 'foo'

# unstage object
assert extract(baz_relative, '0').obj == 'baz'
assert extract(baz_relative, '1/0').obj == 'bar'
assert extract(baz_relative, '2/highly/nested/objects').obj is True
```

1.3.1 API

`pointer.`**`extract`**`(`*obj*, *pointer*, *bypass_ref=False*`)`

Extract member or element of obj according to pointer.

Parameters

- **`obj`** – the object source
- **`pointer`** (*Pointer, str*) – the pointer
- **`bypass_ref`** (*boolean*) – bypass JSON Reference event

class `pointer.`**`DocumentPointer`**`(`*pointer*`)`

Defines a document pointer

Variables

- **`document`** – document name
- **`pointer`** – pointer

`endswith``(`*txt*`)`

used by os.path.join

`extract``(`*obj*, *bypass_ref=False*`)`

Extract subelement from obj, according to pointer. It assums that document is the object.

Parameters

- **`obj`** – the object source
- **`bypass_ref`** – disable JSON Reference errors

`is_inner``()`

Tells if pointer refers to an inner document

class `pointer.`**`Pointer`**`(`*pointer*`)`

Defines a pointer

Variables **`tokens`** – list of PointerToken

extract (*obj*, *bypass_ref=False*)
Extract subelement from obj, according to tokens.

Parameters

- **obj** – the object source
- **bypass_ref** – disable JSON Reference errors

parse (*pointer*)
parse pointer into tokens

class `pointer.`**PointerToken**
A single token

extract (*obj*, *bypass_ref=False*)
Extract parents or subelement from obj, according to current token.

Parameters

- **obj** – the object source
- **bypass_ref** – disable JSON Reference errors

`pointer.`**stage** (*obj*, *parent=None*, *member=None*)
Prepare obj to be staged.

This is almost used for relative JSON Pointers.

Exceptions

class `pointer.`**ExtractError** (*obj*, **args*)
Raised for any errors.

Variables **obj** – the object that raised this event

class `pointer.`**RefError** (*obj*, **args*)
Raised when encoutered a JSON Ref.

Variables **obj** – the object that raised this event

class `pointer.`**LastElement** (*obj*, **args*)
Raised when refers to the last element of a sequence.

Variables **obj** – the object that raised this event

class `pointer.`**OutOfBounds** (*obj*, **args*)
Raised when a member of a mapping does not exists.

Variables **obj** – the object that raised this event

class `pointer.`**OutOfRange** (*obj*, **args*)
Raised when an element of a sequence does not exists.

Variables **obj** – the object that raised this event

1.4 JSON Reference

JSON Reference allows a JSON value to reference another value in a JSON document. This module implements utilities for exploring these objects.

Note: A JSON Reference is a mapping with a unique key `$ref`, which value is a JSON Pointer. For example, this object:

```
{
  "foo": {"$ref": "#/bar"},
  "bar": true
}
```

Can be resolved as:

```
{
  "foo": true,
  "bar": true
}
```

They are some ways to resolve JSON Reference. The simpliest one:

```
from jsonspec.reference import resolve

obj = {
    'foo': ['bar', {'$ref': '#/sub'}, {'$ref': 'obj2#/sub'}],
    'sub': 'baz'
}

assert 'bar' == resolve(obj, '#/foo/0')
assert 'baz' == resolve(obj, '#/foo/1')
assert 'quux' == resolve(obj, '#/foo/2', {
    'obj2': {'sub': 'quux'}
})
```

You may do not already know which documents you will need to resolve your document. For this case, you can plug providers. Actually, these are:

provider	usage
PkgProvider	load any provider from setuptools entrypoints
FilesystemProvider	load documents from filesystem
SpecProvider	load latest JSON Schema specs.

For example, your document refer to stored documents on your filesystem:

```
from jsonspec.reference import Registry
from jsonspec.reference.providers import FileSystemProvider

obj = {
    'foo': {'$ref': 'my:doc#/sub'}
}
provider = FileSystemProvider('/path/to/my/doc', prefix='my:doc')

resolve(obj, '#/foo/2', {
    'obj2': {'sub': 'quux'}
})
```

1.4.1 API

`reference.`**`resolve`**(*obj*, *pointer*, *registry=None*)

resolve a local object

Parameters

- **`obj`** – the local object.

- **`pointer`** (*DocumentPointer, str*) – the pointer
- **`registry`** (*Provider, dict*) – the registry. It mays be omited if inner json references document don't refer to other documents.

Warning: Once pointer is extracted, it won't follow sub mapping /element! For example, the value of:

```
value = resolve({
    'foo': {'$ref': '#/bar'},
    'bar': [{'$ref': '#/baz'}],
    'baz': 'quux',
}, '#/foo')
```

is:

```
assert value == [{'$ref': '#/baz'}]
```

and not:

```
assert value == ['quux']
```

class `reference.`**`Registry`**(*provider=None*)
Register all documents.

Variables

- **`provider`** – all documents
- **`provider`** – Provider, dict

`resolve`(*pointer*)
Resolve from documents.

Parameters **`pointer`** (DocumentPointer) – foo

class `reference.`**`LocalRegistry`**(*doc*, *provider=None*)
Scoped registry to a local document.

Variables

- **`doc`** – the local document
- **`provider`** – all documents
- **`provider`** – Provider, dict
- **`key`** – current document identifier

1.4.2 Utils

jsonspec.reference.util

`reference.util.`**`ref`**(*obj*)
Extracts $ref of object.

class `reference.util.`**`Mapping`**
A Mapping is a generic container for associating key/value pairs.

This class provides concrete generic implementations of all methods except for __getitem__, __iter__, and __len__.

get (*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items () → list of D's (key, value) pairs, as 2-tuples

iteritems () → an iterator over the (key, value) items of D

iterkeys () → an iterator over the keys of D

itervalues () → an iterator over the values of D

keys () → list of D's keys

values () → list of D's values

class `reference.util.`**MutableMapping**

A MutableMapping is a generic container for associating key/value pairs.

This class provides concrete generic implementations of all methods except for __getitem__, __setitem__, __delitem__, __iter__, and __len__.

clear () → None. Remove all items from D.

pop (*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem () → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

setdefault (*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update ([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

1.4.3 Exceptions

class `reference.`**NotFound**

raises when a document is not found

class `reference.`**Forbidden**

raises when a trying to replace <local> document

1.4.4 Defining providers

class `reference.providers.`**PkgProvider** (*namespace=None*, *configuration=None*)

Autoload providers declared into setuptools `entry_points`.

For example, with this setup.cfg:

```
[entry_points]
jsonspec.reference.contributions =
    spec = jsonspec.misc.providers:SpecProvider
```

class `reference.providers.`**FilesystemProvider** (*directory*, *prefix=None*)

Exposes json documents stored into filesystem.

for example, with `prefix=my:pref:` and `directory=my/directory`, this filesystem will be loaded as:

```
    my/directory/
        foo.json        -> my:pref:foo#
        bar.json        -> my:pref:bar#
        baz/
            quux.json   -> my:pref:baz/quux#
```

class `reference.providers.`**`SpecProvider`**

Provides specs of http://json-schema.org/

1.5 Operations

Operations are inspired by the JSON Patch specification.

For example:

```
from jsonspec import operations

obj = {
    'foo': {
        'bar': 'baz',
        'waldo': 'fred'
    },
    'qux': {
        'corge': 'grault'
    }
}
assert operations.move(obj, '/qux/thud', '/foo/waldo') == {
    'foo': {
        'bar': 'baz'
    },
    'qux': {
        'corge': 'grault',
        'thud': 'fred'
    }
}
```

Sources, destinations and locations are expressed with *`pointer.Pointer`*.

1.5.1 Operations

add The `add` operation performs one of the following functions, depending upon what the target location references:

- If the target location specifies an array index, a new value is inserted into the array at the specified index.
- If the target location specifies an object member that does not already exist, a new member is added to the object.
- If the target location specifies an object member that does exist, that member's value is replaced.

For example:

```
        # add or replace a mapping
        operations.add({'foo': 'bar'}, '/baz', 'qux') == {
            'baz': 'qux',
            'foo': 'bar'
```

```
}

# add into a sequence
operations.add(['foo', 'bar'], '/1', 'qux') == ['foo', 'qux', 'bar']
```

remove The `remove` operation removes the value at the target location:

```
# remove a mapping member
operations.remove({
    'baz': 'qux',
    'foo': 'bar'
}, '/baz') == {'foo': 'bar'}

# remove a sequence element
operations.remove(['bar', 'qux', 'baz'], '/1') == ['bar', 'baz']
```

replace The `replace` operation replaces the value at the target location with a new value:

```
operations.replace({
    'baz': 'qux',
    'foo': 'bar'
}, '/baz', 'boo') == {
    'baz': 'boo',
    'foo': 'bar'
}
```

move The `move` operation removes the value at a specified location and adds it to the target location:

```
# move a value into a mapping
operations.move({
    'foo': {
        'bar': 'baz',
        'waldo': 'fred'
    },
    'qux': {
        'corge': 'grault'
    }
}, '/qux/thud', '/foo/waldo') == {
    'foo': {
        'bar': 'baz'
    },
    'qux': {
        'corge': 'grault',
        'thud': 'fred'
    }
}

# move an array element
operations.move([
    'all', 'grass', 'cows', 'eat'
], '/3', '/1') == [
    'all', 'cows', 'eat', 'grass'
]
```

copy The `copy` operation copies the value at a specified location to the target location:

```
operations.copy({
    'foo': {'bar': 42},
}, 'baz', '/foo/bar') == {
        'foo': {'bar': 42}, 'baz': 42
    }
```

check The `test` operation tests that a value at the target location is equal to a specified value:

```
    # testing a value with success
    obj = {
        'baz': 'qux',
        'foo': ['a', 2, 'c']
    }
    assert operations.check(obj, '/baz', 'qux')
    assert operations.check(obj, '/foo/1', 2)

    # testing a value with error
    assert not operations.check({'baz': 'qux'}, '/baz', 'bar')
```

1.5.2 API

`operations.`**`check`**(*doc*, *pointer*, *expected*, *raise_onerror=False*)

Check if value exists into object.

Parameters

- **`doc`** – the document base
- **`pointer`** – the path to search in
- **`expected`** – the expected value
- **`raise_onerror`** – should raise on error?

Returns boolean

`operations.`**`remove`**(*doc*, *pointer*)

Remove element from sequence, member from mapping.

Parameters

- **`doc`** – the document base
- **`pointer`** – the path to search in

Returns the new object

`operations.`**`add`**(*doc*, *pointer*, *value*)

Add element to sequence, member to mapping.

Parameters

- **`doc`** – the document base
- **`pointer`** – the path to add in it
- **`value`** – the new value

Returns the new object

`operations.`**`replace`**(*doc*, *pointer*, *value*)

Replace element from sequence, member from mapping.

Parameters

- **`doc`** – the document base

- **pointer** – the path to search in
- **value** – the new value

Returns the new object

Note: This operation is functionally identical to a "remove" operation for a value, followed immediately by an "add" operation at the same location with the replacement value.

operations.**move**(*doc*, *dest*, *src*)

Move element from sequence, member from mapping.

Parameters

- **doc** – the document base
- **dest** (Pointer) – the destination
- **src** (Pointer) – the source

Returns the new object

Note: it delete then it add to the new location soo the dest must refer to the middle object.

operations.**copy**(*doc*, *dest*, *src*)

Copy element from sequence, member from mapping.

Parameters

- **doc** – the document base
- **dest** (Pointer) – the destination
- **src** (Pointer) – the source

Returns the new object

class operations.**Target**(*document*)

Variables **document** – the document base

add(*pointer*, *value*)

Add element to sequence, member to mapping.

Parameters

- **pointer** – the path to add in it
- **value** – the new value

Returns resolved document

Return type *Target*

The pointer must reference one of:

- The root of the target document - whereupon the specified value becomes the entire content of the target document.
- A member to add to an existing mapping - whereupon the supplied value is added to that mapping at the indicated location. If the member already exists, it is replaced by the specified value.
- An element to add to an existing sequence - whereupon the supplied value is added to the sequence at the indicated location. Any elements at or above the specified index are shifted one position to the right. The specified index must no be greater than the number of elements in the sequence. If the "-"

character is used to index the end of the sequence, this has the effect of appending the value to the sequence.

check (*pointer*, *expected*, *raise_onerror=False*)
Check if value exists into object.

Parameters

- **pointer** – the path to search in
- **expected** – the expected value
- **raise_onerror** – should raise on error?

Returns boolean

copy (*dest*, *src*)
Copy element from sequence, member from mapping.

Parameters

- **dest** (Pointer) – the destination
- **src** (Pointer) – the source

Returns resolved document

Return type *Target*

move (*dest*, *src*)
Move element from sequence, member from mapping.

Parameters

- **dest** (Pointer) – the destination
- **src** (Pointer) – the source

Returns resolved document

Return type *Target*

Note: This operation is functionally identical to a "remove" operation on the "from" location, followed immediately by an "add" operation at the target location with the value that was just removed.

The "from" location MUST NOT be a proper prefix of the "path" location; i.e., a location cannot be moved into one of its children

remove (*pointer*)
Remove element from sequence, member from mapping.

Parameters **pointer** – the path to search in

Returns resolved document

Return type *Target*

replace (*pointer*, *value*)
Replace element from sequence, member from mapping.

Parameters

- **pointer** – the path to search in
- **value** – the new value

Returns resolved document

Return type *Target*

1.6 Validators

This module implements JSON Schema draft03 and draft04.

1.6.1 Basic

```
from jsonspec.validators import load

# data will validate against this schema
validator = load({
    'title': 'Example Schema',
    'type': 'object',
    'properties': {
        'age': {
            'description': 'Age in years',
            'minimum': 0,
            'type': 'integer'
        },
        'firstName': {
            'type': 'string'
        },
        'lastName': {
            'type': 'string'
        }
    },
    'required': [
        'firstName',
        'lastName'
    ]
})

# validate this data
validator.validate({
    'firstName': 'John',
    'lastName': 'Noone',
    'age': 33,
})
```

Choose specification

Schemas will be parsed by the draft04 specification by default. You can setup, or even better mix between draft03 and draft04.

Show these examples:

```
validator = load({
    'id': 'foo',
    'properties': {
        'bar': {
            'id': 'baz'
        },
    },
})
```

foo schema parsed with draft04

baz schema parsed with draft04

```
validator = load({
    'id': 'foo',
    'properties': {
        'bar': {
            'id': 'baz'
        },
    },
}, spec='http://json-schema.org/draft-03/schema#')
```

foo schema parsed with draft03

baz schema parsed with draft03

```
validator = load({
    'id': 'foo',
    'properties': {
        'bar': {
            '$schema': 'http://json-schema.org/draft-03/schema#',
            'id': 'baz'
        },
    },
})
```

foo schema parsed with draft04

baz schema parsed with draft03

About format

This module implements a lot of formats, exposed to every draft:

name	description	enabling
email	validate email	
hostname	validate hostname	
ipv4	validate ipv4	pip install json-spec[ip]
ipv6	validate ipv6	pip install json-spec[ip]
regex	validate regex	
uri	validate uri	
css.color	validate css color	
rfc3339.datetime	see rfc3339	
utc.datetime	YYYY-MM-ddThh:mm:SSZ	
utc.date	YYYY-MM-dd	
utc.time	hh:mm:SS	
utc.millisec	any integer, float	

Some formats rely on external modules, and they are not enabled by default.

Each draft validator aliases they formats to these formats. See *`draft04`* and *`draft03`* methods for more details.

Regarding your needs, you can register your own formats. Use `entry_points` in your `setup.py`. for example:

```
[entry_points]

jsonspec.validators.formats =
    my:format = my.module:validate_format
```

1.6.2 API

`validators.load`(*schema*, *uri=None*, *spec=None*, *provider=None*)

Scaffold a validator against a schema.

Parameters

- **schema** (Mapping) – the schema to compile into a Validator
- **uri** (*Pointer, str*) – the uri of the schema. it may be ignored in case of not cross referencing.
- **spec** (*str*) – fallback to this spec if the schema does not provides ts own
- **provider** (*Mapping, Provider...*) – the other schemas, in case of cross referencing

`validators.draft04.compile`(*schema*, *pointer*, *context*, *scope=None*)

Compiles schema with JSON Schema draft-04.

Parameters

- **schema** (Mapping) – obj to compile
- **pointer** (*Pointer, str*) – uri of the schema
- **context** (Context) – context of this schema

`validators.register`(*compiler=None*, *spec=None*)

Expose compiler to factory.

Parameters

- **compiler** (*callable*) – the callable to expose
- **spec** (*str*) – name of the spec

It can be used as a decorator:

```
@register(spec='my:first:spec')
def my_compiler(schema, pointer, context):
    return Validator(schema)
```

or as a function:

```
def my_compiler(schema, pointer, context):
    return Validator(schema)

register(my_compiler, 'my:second:spec')
```

class `validators.ReferenceValidator`(*pointer*, *context*)

Reference a validator to his pointer.

Variables

- **pointer** – the pointer to the validator
- **context** – the context object
- **default** – return the default validator

- **validator** – return the lazy loaded validator

```
>>> validator = ReferenceValidator('http://json-schema.org/geo#', context)
>>> assert validator({
>>>     'latitude': 0.0124,
>>>     'longitude': 1.2345
>>> })
```

validate (*obj*, *pointer=None*)
Validate object against validator.

Parameters

- **obj** – the object to validate
- **pointer** – the object pointer

class validators.**Draft03Validator** (*attrs*, *uri=None*, *formats=None*)
Implements JSON Schema draft-03 validation.

Variables

- **attrs** – attributes to validate against
- **uri** – uri of the current validator
- **formats** – mapping of available formats

```
>>> validator = Draft03Validator({'min_length': 4})
>>> assert validator('this is sparta')
```

fail (*reason*, *obj*, *pointer=None*)
Called when validation fails.

has_default ()
docstring for has_default

is_optional ()
True by default.

validate (*obj*, *pointer=None*)
Validate object against validator

Parameters **obj** – the object to validate

validate_format (*obj*, *pointer=None*)

Expected draft03	Alias of
color	css.color
date-time	utc.datetime
date	utc.date
time	utc.time
utc-millisec	utc.millisec
regex	regex
style	css.style
phone	phone
uri	uri
email	email
ip-address	ipv4
ipv6	ipv6
host-name	hostname

class `validators.`**`Draft04Validator`**(*attrs*, *uri=None*, *formats=None*)
Implements JSON Schema draft-04 validation.

Variables

- **`attrs`** – attributes to validate against
- **`uri`** – uri of the current validator
- **`formats`** – mapping of available formats

```
>>> validator = Draft04Validator({'min_length': 4})
>>> assert validator('this is sparta')
```

`fail`(*reason*, *obj*, *pointer=None*)
Called when validation fails.

`is_optional`()
Returns True, beceause it is meaningless in draft04.

`validate`(*obj*, *pointer=None*)
Validate object against validator

Parameters **`obj`** – the object to validate

`validate_format`(*obj*, *pointer=None*)

Expected draft04	Alias of
date-time	rfc3339.datetime
email	email
hostname	hostname
ipv4	ipv4
ipv6	ipv6
uri	uri

class `validators.`**`Context`**(*factory*, *registry*, *spec=None*, *formats=None*)

Variables

- **`factory`** – global factory
- **`registry`** – the current registry
- **`spec`** – the current spec
- **`formats`** – the current formats exposed

class `validators.`**`Factory`**(*provider=None*, *spec=None*, *formats=None*)

Variables

- **`provider`** – global registry
- **`spec`** – default spec

1.6.3 Exceptions

class `validators.`**`CompilationError`**(*message*, *schema*)
Raised while schema parsing

class `validators.`**`ReferenceError`**(**args*)
Raised while reference error

class `validators.`**`ValidationError`**(*reason*, *obj=None*, *pointer=None*, *errors=None*)
Raised when validation fails

flatten()

Flatten nested errors.

{pointer: reasons}

CHAPTER 2

Additional Information

2.1 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

2.1.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/johnnoone/json-spec/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

Write Documentation

JSON Spec could always use more documentation, whether as part of the official JSON Spec docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/johnnoone/json-spec/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

2.1.2 Get Started!

Ready to contribute? Here's how to set up *jsonspec* for local development.

1. Fork the *jsonspec* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/jsonspec.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv jsonspec
$ cd jsonspec/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 jsonspec tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

2.1.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.

3. The pull request should work for Python 2.7, 3.3 and for PyPy. Check https://travis-ci.org/johnnoone/json-spec/pull_requests and make sure that the tests pass for all supported Python versions.

2.1.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_jsonspec
```

2.2 Credits

2.2.1 Development Lead

- Xavier Barbosa <clint.northwood@gmail.com>

2.2.2 Contributors

```
Xavier Barbosa <clint.northwood@gmail.com>
Xavier Barbosa <xavier.barbosa@iscool-e.com>
johnnoone <clint.northwood@gmail.com>
```

2.3 History

If you can't find the information you're looking for, have a look at the index or try to find it using the search function:

- genindex
- search