python-json-patch Documentation

Release 1.15

Stefan Kögl

Jun 15, 2017

Contents

python-json-patch is a Python library for applying JSON patches (RFC 6902). Python 2.6, 2.7, 3.2, 3.3 and PyPy are supported.

Contents

CHAPTER 1

Tutorial

Please refer to RFC 6902 for the exact patch syntax.

Creating a Patch

Patches can be created in two ways. One way is to explicitly create a `JsonPatch` object from a list of operations. For convenience, the method `JsonPatch.from_string()` accepts a string, parses it and constructs the patch object from it.

```
>>> import jsonpatch
>>> patch = jsonpatch.JsonPatch([
    {'op': 'add', 'path': '/foo', 'value': 'bar'},
    {'op': 'add', 'path': '/baz', 'value': [1, 2, 3]},
    {'op': 'remove', 'path': '/baz/1'},
    {'op': 'test', 'path': '/baz', 'value': [1, 3]},
    {'op': 'replace', 'path': '/baz/0', 'value': 42},
    {'op': 'remove', 'path': '/baz/1'},
])

# or equivalently
>>> patch = jsonpatch.JsonPatch.from_string('[{"op": "add", ....}]')
```

Another way is to *diff* two objects.

```
>>> src = {'foo': 'bar', 'numbers': [1, 3, 4, 8]}
>>> dst = {'baz': 'qux', 'numbers': [1, 4, 7]}
>>> patch = jsonpatch.JsonPatch.from_diff(src, dst)

# or equivalently
>>> patch = jsonpatch.make_patch(src, dst)
```

Applying a Patch

A patch is always applied to an object.

```
>>> doc = {}
>>> result = patch.apply(doc)
{'foo': 'bar', 'baz': [42]}
```

The `apply` method returns a new object as a result. If `in_place=True` the object is modified in place.

If a patch is only used once, it is not necessary to create a patch object explicitly.

```
>>> obj = {'foo': 'bar'}

# from a patch string
>>> patch = '[{"op": "add", "path": "/baz", "value": "qux"}]'
>>> res = jsonpatch.apply_patch(obj, patch)

# or from a list
>>> patch = [{'op': 'add', 'path': '/baz', 'value': 'qux'}]
>>> res = jsonpatch.apply_patch(obj, patch)
```

CHAPTER 2

The `jsonpatch` module

Apply JSON-Patches (RFC 6902)

class `jsonpatch.`**`AddOperation`**(*operation*)
Adds an object property or an array element.

class `jsonpatch.`**`CopyOperation`**(*operation*)
Copies an object property or an array element to a new location

exception `jsonpatch.`**`InvalidJsonPatch`**
Raised if an invalid JSON Patch is created

class `jsonpatch.`**`JsonPatch`**(*patch*)
A JSON Patch is a list of Patch Operations.

```
>>> patch = JsonPatch([
...     {'op': 'add', 'path': '/foo', 'value': 'bar'},
...     {'op': 'add', 'path': '/baz', 'value': [1, 2, 3]},
...     {'op': 'remove', 'path': '/baz/1'},
...     {'op': 'test', 'path': '/baz', 'value': [1, 3]},
...     {'op': 'replace', 'path': '/baz/0', 'value': 42},
...     {'op': 'remove', 'path': '/baz/1'},
... ])
>>> doc = {}
>>> result = patch.apply(doc)
>>> expected = {'foo': 'bar', 'baz': [42]}
>>> result == expected
True
```

JsonPatch object is iterable, so you could easily access to each patch statement in loop:

```
>>> lpatch = list(patch)
>>> expected = {'op': 'add', 'path': '/foo', 'value': 'bar'}
>>> lpatch[0] == expected
True
>>> lpatch == patch.patch
True
```

Also JsonPatch could be converted directly to `bool` if it contains any operation statements:

```
>>> bool(patch)
True
>>> bool(JsonPatch([]))
False
```

This behavior is very handy with *make_patch()* to write more readable code:

```
>>> old = {'foo': 'bar', 'numbers': [1, 3, 4, 8]}
>>> new = {'baz': 'qux', 'numbers': [1, 4, 7]}
>>> patch = make_patch(old, new)
>>> if patch:
...     # document have changed, do something useful
...     patch.apply(old)
{...}
```

apply(*obj*, *in_place=False*)

Applies the patch to given object.

Parameters

- **obj** (*dict*) – Document object.
- **in_place** (*bool*) – Tweaks way how patch would be applied - directly to specified *obj* or to his copy.

Returns Modified *obj*.

classmethod from_diff(*src*, *dst*, *optimization=True*)

Creates JsonPatch instance based on comparing of two document objects. Json patch would be created for *src* argument against *dst* one.

Parameters

- **src** (*dict*) – Data source document object.
- **dst** (*dict*) – Data source document object.

Returns *JsonPatch* instance.

```
>>> src = {'foo': 'bar', 'numbers': [1, 3, 4, 8]}
>>> dst = {'baz': 'qux', 'numbers': [1, 4, 7]}
>>> patch = JsonPatch.from_diff(src, dst)
>>> new = patch.apply(src)
>>> new == dst
True
```

classmethod from_string(*patch_str*)

Creates JsonPatch instance from string source.

Parameters **patch_str** (*str*) – JSON patch as raw string.

Returns *JsonPatch* instance.

to_string()

Returns patch set as JSON string.

exception jsonpatch.**JsonPatchConflict**

Raised if patch could not be applied due to conflict situation such as: - attempt to add object key then it already exists; - attempt to operate with nonexistence object key; - attempt to insert value to array at position beyond of it size; - etc.

exception jsonpatch.**JsonPatchException**
Base Json Patch exception

exception jsonpatch.**JsonPatchTestFailed**
A Test operation failed

class jsonpatch.**MoveOperation**(*operation*)
Moves an object property or an array element to new location.

class jsonpatch.**PatchOperation**(*operation*)
A single operation inside a JSON Patch.

apply(*obj*)
Abstract method that applies patch operation to specified object.

class jsonpatch.**RemoveOperation**(*operation*)
Removes an object property or an array element.

class jsonpatch.**ReplaceOperation**(*operation*)
Replaces an object property or an array element by new value.

class jsonpatch.**TestOperation**(*operation*)
Test value by specified location.

jsonpatch.**apply_patch**(*doc*, *patch*, *in_place=False*)
Apply list of patches to specified json document.

Parameters

- **doc** (*dict*) – Document object.
- **patch** (*list or str*) – JSON patch as list of dicts or raw JSON-encoded string.
- **in_place** (*bool*) – While `True` patch will modify target document. By default patch will be applied to document copy.

Returns Patched document object.

Return type dict

```
>>> doc = {'foo': 'bar'}
>>> patch = [{'op': 'add', 'path': '/baz', 'value': 'qux'}]
>>> other = apply_patch(doc, patch)
>>> doc is not other
True
>>> other == {'foo': 'bar', 'baz': 'qux'}
True
>>> patch = [{'op': 'add', 'path': '/baz', 'value': 'qux'}]
>>> apply_patch(doc, patch, in_place=True) == {'foo': 'bar', 'baz': 'qux'}
True
>>> doc == other
True
```

jsonpatch.**get_loadjson**()
adds the object_pairs_hook parameter to json.load when possible

The "object_pairs_hook" parameter is used to handle duplicate keys when loading a JSON object. This parameter does not exist in Python 2.6. This methods returns an unmodified json.load for Python 2.6 and a partial function with object_pairs_hook set to multidict for Python versions that support the parameter.

jsonpatch.**make_patch**(*src*, *dst*)
Generates patch by comparing of two document objects. Actually is a proxy to *JsonPatch.from_diff()* method.

Parameters

- **src** (*dict*) – Data source document object.
- **dst** (*dict*) – Data source document object.

```
>>> src = {'foo': 'bar', 'numbers': [1, 3, 4, 8]}
>>> dst = {'baz': 'qux', 'numbers': [1, 4, 7]}
>>> patch = make_patch(src, dst)
>>> new = patch.apply(src)
>>> new == dst
True
```

jsonpatch.**multidict**(*ordered_pairs*)

Convert duplicate keys values to lists.

CHAPTER 3

Commandline Utilities

The JSON patch package contains the commandline utilities `jsondiff` and `jsonpatch`.

jsondiff

The program `jsondiff` can be used to create a JSON patch by comparing two JSON files

```
usage: jsondiff [-h] [--indent INDENT] [-v] FILE1 FILE2

Diff two JSON files

positional arguments:
  FILE1
  FILE2

optional arguments:
  -h, --help       show this help message and exit
  --indent INDENT  Indent output by n spaces
  -v, --version    show program's version number and exit
```

Example

```
# inspect JSON files
$ cat a.json
{ "a": [1, 2], "b": 0 }

$ cat b.json
{ "a": [1, 2, 3], "c": 100 }

# show patch in "dense" representation
$ jsondiff a.json b.json
[{"path": "/a/2", "value": 3, "op": "add"}, {"path": "/b", "op": "remove"}, {"path":
↪"/c", "value": 100, "op": "add"}]
```

```
# show patch with some indentation
$ jsondiff a.json b.json --indent=2
[
  {
    "path": "/a/2",
    "value": 3,
    "op": "add"
  },
  {
    "path": "/b",
    "op": "remove"
  },
  {
    "path": "/c",
    "value": 100,
    "op": "add"
  }
]
```

jsonpatch

The program `jsonpatch` is used to apply JSON patches on JSON files.

```
usage: jsonpatch [-h] [--indent INDENT] [-v] ORIGINAL PATCH

Apply a JSON patch on a JSON files

positional arguments:
  ORIGINAL         Original file
  PATCH            Patch file

optional arguments:
  -h, --help       show this help message and exit
  --indent INDENT  Indent output by n spaces
  -v, --version    show program's version number and exit
```

Example

```
# create a patch
$ jsondiff a.json b.json > patch.json

# show the result after applying a patch
$ jsonpatch a.json patch.json
{"a": [1, 2, 3], "c": 100}

$ jsonpatch a.json patch.json --indent=2
{
  "a": [
    1,
    2,
    3
  ],
```

```
  "c": 100
}

# pipe result into new file
$ jsonpatch a.json patch.json --indent=2 > c.json

# c.json now equals b.json
$ jsondiff b.json c.json
[]
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

j

Index