## Contents

1 Tutorial 3  
  1.1 Creating a Patch 3  
  1.2 Applying a Patch 4  
  1.3 Dealing with Custom Types 4  

2 The jsonpatch module 7  

3 Commandline Utilities 9  
  3.1 jsondiff 9  
  3.2 jsonpatch 10  

4 Indices and tables 13  

Python Module Index 15  

Index 17
python-json-patch is a Python library for applying JSON patches (RFC 6902). Python 2.7 and 3.4+ are supported. Tests are run on both CPython and PyPy.

Contents
Please refer to RFC 6902 for the exact patch syntax.

### 1.1 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.

```python
>>> 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.

```python
>>> 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)
```
1.2 Applying a Patch

A patch is always applied to an object.

```python
>>> 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.

```python
>>> 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)
```

1.3 Dealing with Custom Types

Custom JSON dump and load functions can be used to support custom types such as `decimal.Decimal`. The following examples shows how the `simplejson` package, which has native support for Python's `Decimal` type, can be used to create a custom `JsonPatch` subclass with `Decimal` support:

```python
>>> import decimal
>>> import simplejson

>>> class DecimalJsonPatch(jsonpatch.JsonPatch):
    @staticmethod
    def json_dumper(obj):
        return simplejson.dumps(obj)

    @staticmethod
    def json_loader(obj):
        return simplejson.loads(obj, use_decimal=True,
                                object_pairs_hook=jsonpatch.multidict)

>>> src = {}
>>> dst = {'bar': decimal.Decimal('1.10')}
>>> patch = DecimalJsonPatch.from_diff(src, dst)
>>> doc = {'foo': 1}
>>> result = patch.apply(doc)
{'foo': 1, 'bar': Decimal('1.10')}
```

Instead of subclassing it is also possible to pass a dump function to `from_diff`:

```python
>>> patch = jsonpatch.JsonPatch.from_diff(src, dst, dumps=simplejson.dumps)
```

a dumps function to `to_string`:
and load function to `from_string`:

```python
>>> import functools
>>> loads = functools.partial(simplejson.loads, use_decimal=True,
                          object_pairs_hook=jsonpatch.multidict)
>>> patch.from_string(serialized_patch, loads=loads)
>>> doc = {'foo': 1}
>>> result = patch.apply(doc)
{'foo': 1, 'bar': Decimal('1.10')}
```
CHAPTER 2

The jsonpatch module

Apply JSON-Patches (RFC 6902)

```python
class jsonpatch.AddOperation(operation, pointer_cls=<class 'jsonpointer.JsonPointer'>):
    Adds an object property or an array element.

class jsonpatch.CopyOperation(operation, pointer_cls=<class 'jsonpointer.JsonPointer'>):
    Copies an object property or an array element to a new location

eexception jsonpatch.InvalidJsonPatch
    Raised if an invalid JSON Patch is created

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

eexception jsonpatch.JsonPatchException
    Base Json Patch exception

eexception jsonpatch.JsonPatchTestFailed
    A Test operation failed

class jsonpatch.MoveOperation(operation, pointer_cls=<class 'jsonpointer.JsonPointer'>):
    Moves an object property or an array element to a new location.

class jsonpatch.PatchOperation(operation, pointer_cls=<class 'jsonpointer.JsonPointer'>):
    A single operation inside a JSON Patch.

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

class jsonpatch.RemoveOperation(operation, pointer_cls=<class 'jsonpointer.JsonPointer'>):
    Removes an object property or an array element.

class jsonpatch.ReplaceOperation(operation, pointer_cls=<class 'jsonpointer.JsonPointer'>):
    Replaces an object property or an array element by a new value.
```
class jsonpatch.TestOperation(operation, pointer_cls=<class 'jsonpointer.JsonPointer'>)
    Test value by specified location.

jsonpatch.apply_patch(doc, patch, in_place=False, pointer_cls=<class 'jsonpointer.JsonPointer'>)
    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.
* pointer_cls (Type[JsonPointer]) – JSON pointer class to use.

Returns Patched document object.

Return type dict

```python
>>> doc = {'foo': 'bar'}
>>> patch = [{'op': 'add', 'path': '/baz', 'value': 'qux'}]
>>> other = apply_patch(doc, patch)
>>> other 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.make_patch(src, dst, pointer_cls=<class 'jsonpointer.JsonPointer'>)
Generates patch by comparing 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.
* pointer_cls (Type[JsonPointer]) – JSON pointer class to use.

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

### 3.1 jsondiff

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

```

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
  -u, --preserve-unicode Output Unicode character as-is without using Code Point
  -v, --version        show program's version number and exit
```

#### 3.1.1 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
```

(continues on next page)
3.2 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
  -b, --backup    Back up ORIGINAL if modifying in-place
  -i, --in-place  Modify ORIGINAL in-place instead of to stdout
  -v, --version   show program's version number and exit
  -u, --preserve-unicode Output Unicode character as-is without using Code Point

3.2.1 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
  jsonpatch, 7
Index

A
AddOperation (class in jsonpatch), 7
apply() (jsonpatch.PatchOperation method), 7
apply_patch() (in module jsonpatch), 8

C
CopyOperation (class in jsonpatch), 7

I
InvalidJsonPatch, 7

J
jsonpatch (module), 7
JsonPatchConflict, 7
JsonPatchException, 7
JsonPatchTestFailed, 7

M
make_patch() (in module jsonpatch), 8
MoveOperation (class in jsonpatch), 7
multidict() (in module jsonpatch), 8

P
PatchOperation (class in jsonpatch), 7

R
RemoveOperation (class in jsonpatch), 7
ReplaceOperation (class in jsonpatch), 7

T
TestOperation (class in jsonpatch), 7