# python-json-pointer Documentation

*Release 2.3*

**Stefan Kögl**

**Dec 12, 2022**

# Contents

*python-json-pointer* is a Python library for resolving JSON pointers (RFC 6901). Python 2.7, 3.4+ and PyPy are supported.

**Contents**

# CHAPTER 1

# Tutorial

Please refer to RFC 6901 for the exact pointer syntax. `jsonpointer` has two interfaces. The `resolve_pointer` method is basically a deep `get`.

```
>>> from jsonpointer import resolve_pointer
>>> obj = {"foo": {"anArray": [ {"prop": 44}], "another prop": {"baz": "A string" }}}

>>> resolve_pointer(obj, '') == obj
True

>>> resolve_pointer(obj, '/foo') == obj['foo']
True

>>> resolve_pointer(obj, '/foo/another prop') == obj['foo']['another prop']
True

>>> resolve_pointer(obj, '/foo/another prop/baz') == obj['foo']['another prop']['baz']
True

>>> resolve_pointer(obj, '/foo/anArray/0') == obj['foo']['anArray'][0]
True

>>> resolve_pointer(obj, '/some/path', None) == None
True
```

The `set_pointer` method allows modifying a portion of an object using JSON pointer notation:

```
>>> from jsonpointer import set_pointer
>>> obj = {"foo": {"anArray": [ {"prop": 44}], "another prop": {"baz": "A string" }}}

>>> set_pointer(obj, '/foo/anArray/0/prop', 55)
{'foo': {'another prop': {'baz': 'A string'}, 'anArray': [{'prop': 55}]}}

>>> obj
{'foo': {'another prop': {'baz': 'A string'}, 'anArray': [{'prop': 55}]}}
```

By default `set_pointer` modifies the original object. Pass `inplace=False` to create a copy and modify the copy instead:

```
>>> from jsonpointer import set_pointer
>>> obj = {"foo": {"anArray": [ {"prop": 44}], "another prop": {"baz": "A string" }}}
```

```
>>> set_pointer(obj, '/foo/anArray/0/prop', 55, inplace=False)
{'foo': {'another prop': {'baz': 'A string'}, 'anArray': [{'prop': 55}]}}
```

```
>>> obj
{'foo': {'another prop': {'baz': 'A string'}, 'anArray': [{'prop': 44}]}}
```

The `JsonPointer` class wraps a (string) path and can be used to access the same path on several objects.

```
>>> import jsonpointer

>>> pointer = jsonpointer.JsonPointer('/foo/1')

>>> obj1 = {'foo': ['a', 'b', 'c']}
>>> pointer.resolve(obj1)
'b'

>>> obj2 = {'foo': {'0': 1, '1': 10, '2': 100}}
>>> pointer.resolve(obj2)
10
```

CHAPTER 2

# The `jsonpointer` module

Identify specific nodes in a JSON document (RFC 6901)

**class** `jsonpointer.`**EndOfList**(*list_*)
Result of accessing element "-" of a list

**class** `jsonpointer.`**JsonPointer**(*pointer*)
A JSON Pointer that can reference parts of a JSON document

**contains**(*ptr*)
Returns True if self contains the given ptr

**classmethod from_parts**(*parts*)
Constructs a JsonPointer from a list of (unescaped) paths

```
>>> JsonPointer.from_parts(['a', '~', '/', 0]).path == '/a/~0/~1/0'
True
```

**get**(*doc*, *default=<object object>*)
Resolves the pointer against doc and returns the referenced object

**classmethod get_part**(*doc*, *part*)
Returns the next step in the correct type

**get_parts**()
Returns the list of the parts. For example, JsonPointer('/a/b').get_parts() == ['a', 'b']

**join**(*suffix*)
Returns a new JsonPointer with the given suffix append to this ptr

**path**
Returns the string representation of the pointer

```
>>> ptr = JsonPointer('/~0/0/~1').path == '/~0/0/~1'
```

**resolve**(*doc*, *default=<object object>*)
Resolves the pointer against doc and returns the referenced object

**set** (*doc*, *value*, *inplace=True*)
Resolve the pointer against the doc and replace the target with value.

**to_last** (*doc*)
Resolves ptr until the last step, returns (sub-doc, last-step)

**walk** (*doc*, *part*)
Walks one step in doc and returns the referenced part

**exception** jsonpointer.**JsonPointerException**

jsonpointer.**pairwise** (*iterable*)
Transforms a list to a list of tuples of adjacent items

s -> (s0,s1), (s1,s2), (s2, s3), ...

```
>>> list(pairwise([]))
[]
```

```
>>> list(pairwise([1]))
[]
```

```
>>> list(pairwise([1, 2, 3, 4]))
[(1, 2), (2, 3), (3, 4)]
```

jsonpointer.**resolve_pointer** (*doc*, *pointer*, *default=<object object>*)
Resolves pointer against doc and returns the referenced object

```
>>> obj = {'foo': {'anArray': [ {'prop': 44}], 'another prop': {'baz': 'A string'␣
↪}}, 'a%20b': 1, 'c d': 2}
```

```
>>> resolve_pointer(obj, '') == obj
True
```

```
>>> resolve_pointer(obj, '/foo') == obj['foo']
True
```

```
>>> resolve_pointer(obj, '/foo/another prop') == obj['foo']['another prop']
True
```

```
>>> resolve_pointer(obj, '/foo/another prop/baz') == obj['foo']['another prop'][
↪'baz']
True
```

```
>>> resolve_pointer(obj, '/foo/anArray/0') == obj['foo']['anArray'][0]
True
```

```
>>> resolve_pointer(obj, '/some/path', None) == None
True
```

```
>>> resolve_pointer(obj, '/a b', None) == None
True
```

```
>>> resolve_pointer(obj, '/a%20b') == 1
True
```

```
>>> resolve_pointer(obj, '/c d') == 2
True
```

```
>>> resolve_pointer(obj, '/c%20d', None) == None
True
```

jsonpointer.**set_pointer**(*doc*, *pointer*, *value*, *inplace=True*)

Resolves pointer against doc and sets the value of the target within doc.

With inplace set to true, doc is modified as long as pointer is not the root.

```
>>> obj = {'foo': {'anArray': [ {'prop': 44}], 'another prop': {'baz': 'A string'␣
↪}}}
```

```
>>> set_pointer(obj, '/foo/anArray/0/prop', 55) ==     {'foo': {'another prop': {
↪'baz': 'A string'}, 'anArray': [{'prop': 55}]}}
True
```

```
>>> set_pointer(obj, '/foo/yet another prop', 'added prop') ==     {'foo': {
↪'another prop': {'baz': 'A string'}, 'yet another prop': 'added prop', 'anArray
↪': [{'prop': 55}]}}
True
```

```
>>> obj = {'foo': {}}
>>> set_pointer(obj, '/foo/a%20b', 'x') ==     {'foo': {'a%20b': 'x' }}
True
```

# CHAPTER 3

# The `jsonpointer` commandline utility

The JSON pointer package also installs a `jsonpointer` commandline utility that can be used to resolve a JSON pointers on JSON files.

The program has the following usage

```
usage: jsonpointer [-h] [--indent INDENT] [-v] POINTER FILE [FILE ...]

Resolve a JSON pointer on JSON files

positional arguments:
  POINTER          File containing a JSON pointer expression
  FILE             Files for which the pointer should be resolved

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
```

## 3.1 Example

```
# inspect JSON files
$ cat a.json
{ "a": [1, 2, 3] }

$ cat b.json
{ "a": {"b": [1, 3, 4]}, "b": 1 }

# inspect JSON pointer
$ cat ptr.json
"/a"

# resolve JSON pointer
```

(continues on next page)

(continued from previous page)

```
$ jsonpointer ptr.json a.json b.json
[1, 2, 3]
{"b": [1, 3, 4]}
```

CHAPTER 4

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## j

# Index