JSON Patch
draft-ietf-appsawg-json-patch-06

Abstract

   JSON Patch defines the media type "application/json-patch", a JSON
   document structure for expressing a sequence of operations to apply
   to a JSON document.

Status of this Memo

   This Internet-Draft is submitted in full conformance with the
   provisions of BCP 78 and BCP 79.

   Internet-Drafts are working documents of the Internet Engineering
   Task Force (IETF). Note that other groups may also distribute
   working documents as Internet-Drafts. The list of current Internet-
   Drafts is at http://datatracker.ietf.org/drafts/current/.

   Internet-Drafts are draft documents valid for a maximum of six months
   and may be updated, replaced, or obsoleted by other documents at any
   time. It is inappropriate to use Internet-Drafts as reference
   material or to cite them other than as "work in progress."

   This Internet-Draft will expire on April 25, 2013.

Copyright Notice

Table of Contents

1. Introduction

   JavaScript Object Notation (JSON) [RFC4627] is a common format for
   the exchange and storage of structured data. HTTP PATCH [RFC5789]
   extends the Hypertext Transfer Protocol (HTTP) [RFC2616] with a
   method to perform partial modifications to resources.

   JSON Patch is a format (identified by the media type "application/
   json-patch") for expressing a sequence of operations to apply to a
   target JSON document, suitable for use with the HTTP PATCH method.

2. Conventions

   The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
   "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
   document are to be interpreted as described in RFC 2119 [RFC2119].

   See Section 5 for information about handling errors.

3. Document Structure

   A JSON Patch document is a JSON [RFC4627] document whose root is an
   array of objects. Each object represents a single operation to be
   applied to the target JSON document.

   An example JSON Patch document:

```
[
  { "op": "test", "path": "/a/b/c", "value": "foo" },
  { "op": "remove", "path": "/a/b/c" },
  { "op": "add", "path": "/a/b/c", "value": [ "foo", "bar" ] },
  { "op": "replace", "path": "/a/b/c", "value": 42 },
  { "op": "move", "path": "/a/b/c", "to": "/a/b/d" },
  { "op": "copy", "path": "/a/b/d", "to": "/a/b/e" }
]
```

   Evaluation of a JSON Patch document begins with a target JSON
   document. Operations are applied sequentially in the order they
   appear in the array. Each operation in the sequence is applied to
   the target document; the resulting document becomes the target of the
   next operation. Evaluation continues until all operations are
   successfully applied, or an error condition is encountered.

[4](#). Operations

Operation objects MUST have exactly one "op" member, whose value
indicates the operation to perform. Its value MUST be one of "add",
"remove", "replace", "move", "copy" or "test". The semantics of each
is defined below.

Additionally, operation objects MUST have exactly one "path" member,
whose value MUST be a string containing a [JSON-Pointer] value that
references the location within the target document to perform the
operation (the "target location").

Other members of operation objects MUST be ignored, unless they are
explicitly allowed by the definition of the operation.

Note that the ordering of members in JSON objects is not significant;
therefore, the following operations are equivalent:

```
{ "op": "add", "path": "/a/b/c", "value": "foo" }
{ "path": "/a/b/c", "op": "add", "value": "foo" }
{ "value": "foo", "path": "/a/b/c", "op": "add" }
```

[4.1](#). add

The "add" operation adds a new value at the target location. The
operation object MUST contain a "value" member that specifies the
value to be added.

When the operation is applied, the target location MUST reference one
of:

o the root of the target document,

o a member to add to an existing object, or

o an element to add to an existing array.

For example:

```
{ "op": "add", "path": "/a/b/c", "value": [ "foo", "bar" ] }
```

If the target location references an element of an existing array,
any elements at or above the specified index are shifted one position
to the right. The specified index MUST NOT be greater than the
number of elements in the array.

When the "-" character is used to index the end of the array, this
has the effect of appending the value to the array.

Note that this operation will, in common use, have a target location
that does not resolve to an existing value, resulting in the
pointer's error handling algorithm being invoked. This specification
defines the error handling algorithm for "add" pointers to explicitly
ignore the error and perform the operation as specified.

4.2. remove

The "remove" operation removes the value at the target location.

The target location MUST exist for the operation to be successful.

For example:

{ "op": "remove", "path": "/a/b/c" }

If removing an element from an array, any elements above the
specified index are shifted one position to the left.

4.3. replace

The "replace" operation replaces the value at the target location
with a new value. The operation object MUST contain a "value" member
that specifies the replacement value.

The target location MUST exist for the operation to be successful.

For example:

{ "op": "replace", "path": "/a/b/c", "value": 42 }

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

4.4. move

The "move" operation removes the value at the target location and
adds it to another location.

The operation object MUST contain a "to" member, a string containing
a JSON Pointer value that references the location in the target
document to add the value to.

The "to" location MUST reference one of:

o the member to add to an existing object, or

   o an element to add to an existing array.

   For example:

   { "op": "move", "path": "/a/b/c", "to": "/a/b/d" }

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

   The location in the "to" member MUST NOT be part of the location
   defined by "path"; i.e., a location cannot be moved into one of its
   children.

   The location in the "to" member MUST NOT reference a member of an
   existing object in the target document, unless "path" and "to"
   specify the same object, which has no effect.

   If the location in the "to" member references an element of an
   existing array, any elements at or above the specified index are
   shifted one position to the right. The specified index MUST NOT be
   greater than the number of elements in the array.

[4.5](). copy

   The "copy" operation copies the value at the target location to
   another location.

   The operation object MUST contain a "to" member, a string containing
   a JSON Pointer value that references the location in the target
   document to add the value to.

   This location MUST reference one of:

   o the member to add to an existing object, or

   o an element to add to an existing array.

   For example:

   { "op": "copy", "path": "/a/b/c", "to": "/a/b/e" }

   The location in the "to" member MUST NOT be part of the location
   defined by "path"; i.e., a location cannot be copied into one of its
   children.

   The location in the "to" member MUST NOT reference a member of an
   existing object in the target document, unless "path" and "to"

   specify the same object, which has no effect.

   If the location in the "to" member references an element of an
   existing array, any elements at or above the specified index are
   shifted one position to the right. The specified index MUST NOT be
   greater than the number of elements in the array.

[4.6](#). test

   The "test" operation tests that a value at the target location is
   equal to a specified value.

   The operation object MUST contain a "value" member that conveys the
   value to be compared to that at the target location.

   The target location MUST be equal to the "value" value for the
   operation to be considered successful.

   Here, "equal" means that the value at the target location and the
   value conveyed by "value" are of the same JSON type, and considered
   equal by the following rules for that type:

   o strings: are considered equal if, after unescaping any sequence(s)
      in both strings starting with a reverse solidus, they contain the
      same number of Unicode characters and their code points are
      position-wise equal.

   o numbers: are considered equal if subtracting one from the other
      results in 0.

   o arrays: are considered equal if they contain the same number of
      values, and each value can be considered equal to the value at the
      corresponding position in the other array.

   o objects: are considered equal if they contain the same number of
      members, and each member can be considered equal to a member in
      the other object, by comparing their keys as strings, and values
      using this list of type-specific rules.

   o literals (false, true and null): are considered equal if they are
      the same.

   Note that this is a logical comparison; e.g., whitespace between the
   member values of an array is not significant.

   Also, note that ordering of the serialisation of object members is
   not significant.

   For example:

   { "op": "test", "path": "/a/b/c", "value": "foo" }

5. Error Handling

   If a RFC2119 [RFC2119] requirement is violated by a JSON Patch
   document, or if an operation is not successful, evaluation of the
   JSON Patch document SHOULD terminate and application of the entire
   patch document SHALL NOT be deemed successful.

   See [RFC5789], Section 2.2 for considerations regarding handling
   errors when JSON Patch is used with the HTTP PATCH method, including
   suggested status codes to use to indicate various conditions.

   Note that as per [RFC5789], when used with the PATCH HTTP method, it
   is atomic. Therefore, the following patch would result in no changes
   being made to the document at all (because the "test" operation
   results in an error).

   [
     { "op": "replace", "path": "/a/b/c", "value": 42 },
     { "op": "test", "path": "/a/b/c", "value": "C" }
   ]

6. IANA Considerations

   The Internet media type for a JSON Patch document is application/
   json-patch.

   Type name: application

   Subtype name: json-patch

   Required parameters: none

   Optional parameters: none

   Encoding considerations: binary

   Security considerations:
      See Security Considerations in section 7.

   Interoperability considerations: N/A

   Published specification:
      [this memo]

   Applications that use this media type:
      Applications that manipulate JSON documents.

   Additional information:

      Magic number(s): N/A

      File extension(s): .json-patch

      Macintosh file type code(s): TEXT

   Person & email address to contact for further information:
      Paul C. Bryan <pbryan@anode.ca>

   Intended usage: COMMON

   Restrictions on usage: none

   Author: Paul C. Bryan <pbryan@anode.ca>

   Change controller: IETF

7. Security Considerations

   This specification has the same security considerations as JSON
   [RFC4627] and [JSON-Pointer].

8. Acknowledgements

   The following individuals contributed ideas, feedback and wording to
   this specification:

      Mike Acar, Mike Amundsen, Paul Davis, Murray S. Kucherawy, Dean
      Landolt, Randall Leeds, James Manger, Julian Reschke, James Snell,
      Eli Stevens.

   The structure of a JSON Patch document was influenced by the XML
   Patch document [RFC5261] specification.

9. References

9.1. **Normative References**

[JSON-Pointer]
             Bryan, P. and K. Zyp, "JSON Pointer",
             draft-ietf-appsawg-json-pointer-04 (work in progress),
             March 2012.

[RFC2119]   Bradner, S., "Key words for use in RFCs to Indicate
             Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC4627]   Crockford, D., "The application/json Media Type for
             JavaScript Object Notation (JSON)", RFC 4627, July 2006.

9.2. **Informative References**

[RFC2616]   Fielding, R., Gettys, J., Mogul, J., Frystyk, H.,
             Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext
             Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC5261]   Urpalainen, J., "An Extensible Markup Language (XML) Patch
             Operations Framework Utilizing XML Path Language (XPath)
             Selectors", RFC 5261, September 2008.

[RFC5789]   Dusseault, L. and J. Snell, "PATCH Method for HTTP",
             RFC 5789, March 2010.

Appendix A. **Examples**

A.1. **Adding an Object Member**

An example target JSON document:

```
{
  "foo": "bar"
}
```

A JSON Patch document:

```
[
  { "op": "add", "path": "/baz", "value": "qux" }
]
```

The resulting JSON document:

```
{
  "baz": "qux",
  "foo": "bar"
```

```
      }
```

A.2. **Adding an Array Element**

   An example target JSON document:

```
   {
     "foo": [ "bar", "baz" ]
   }
```

   A JSON Patch document:

```
   [
     { "op": "add", "path": "/foo/1", "value": "qux" }
   ]
```

   The resulting JSON document:

```
   {
     "foo": [ "bar", "qux", "baz" ]
   }
```

A.3. **Removing an Object Member**

   An example target JSON document:

```
   {
     "baz": "qux",
     "foo": "bar"
   }
```

   A JSON Patch document:

```
   [
     { "op": "remove", "path": "/baz" }
   ]
```

   The resulting JSON document:

```
   {
     "foo": "bar"
   }
```

A.4. **Removing an Array Element**

   An example target JSON document:

```
{
  "foo": [ "bar", "qux", "baz" ]
}
```

A JSON Patch document:

```
[
  { "op": "remove", "path": "/foo/1" }
]
```

The resulting JSON document:

```
{
  "foo": [ "bar", "baz" ]
}
```

A.5. Replacing a Value

An example target JSON document:

```
{
  "baz": "qux",
  "foo": "bar"
}
```

A JSON Patch document:

```
[
  { "op": "replace", "path": "/baz", "value": "boo" }
]
```

The resulting JSON document:

```
{
  "baz": "boo",
  "foo": "bar"
}
```

A.6. Moving a Value

An example target JSON document:

```
{
  "foo": {
    "bar": "baz",
    "waldo": "fred"
  }
  "qux": {
    "corge": "grault"
  }
}
```

A JSON Patch document:

```
[
  { "op": "move", "path": "/foo/waldo", to: "/qux/thud" }
]
```

The resulting JSON document:

```
{
  "foo": {
    "bar": "baz"
  }
  "qux": {
    "corge": "grault",
    "thud": "fred"
  }
}
```

A.7. Moving an Array Element

An example target JSON document:

```
{
  "foo": [ "all", "grass", "cows", "eat" ]
}
```

A JSON Patch document:

```
[
  { "op": "move", "path": "/foo/1", "to": "/foo/3" }
]
```

The resulting JSON document:

```
{
  "foo": [ "all", "cows", "eat", "grass" ]
}
```

A.8. Testing a Value: Success

   An example target JSON document:

   {
     "baz": "qux",
     "foo": ["a", 2, "c"]
   }

   A JSON Patch document that will result in successful evaluation:

   [
     { "op": "test", "path": "/baz", "value": "qux" },
     { "op": "test", "path": "/foo/1", "value": 2 }
   ]

A.9. Testing a Value: Error

   An example target JSON document:

   {
     "baz": "qux"
   }

   A JSON Patch document that will result in an error condition:

   [
     { "op": "test", "path": "/baz", "value": "bar" }
   ]

A.10. Adding a nested Member Object

   An example target JSON document:

   {
     "foo": "bar"
   }

   A JSON Patch document:

   [
     { "op": "add", "path": "/child", "value": { "grandchild": { } } }
   ]

   The resulting JSON document:

```
   {
     "foo": "bar",
     "child": {
       "grandchild": {
       }
     }
   }
```

A.11. Ignoring Unrecognized Elements

   An example target JSON document:

```
   {
     "foo":"bar"
   }
```

   A JSON Patch document:

```
   [
     { "op":"add", "path":"/baz", "value":"qux", "xyz":123 }
   ]
```

   The resulting JSON document:

```
   {
     "foo":"bar",
     "baz":"qux"
   }
```

A.12. Adding to a Non-existant Target

   An example target JSON document:

```
   {
     "foo": "bar"
   }
```

   A JSON Patch document:

```
   [
     { "op": "add", "path": "/baz/bat", "value": "qux" }
   ]
```

   This JSON Patch document, applied to the target JSON document above,
   would result in an error (therefore not being applied) because the
   "add" operation's target location that references neither the root of
   the document, nor a member of an existing object, nor a member of an
   existing array.

[A.13](#). Invalid JSON Patch Document

A JSON Patch document:

```
[
  { "op":"add", "path":"/baz", "value":"qux", "op":"remove" }
]
```

This JSON Patch document cannot be treated as an "add" operation
since there is a later "op":"remove" element. A JSON parser that
hides such duplicate element names therefore cannot be used unless it
always exposes only the last element with a given name (eg
"op":"remove" in this example).

Authors' Addresses

Paul C. Bryan (editor)
Salesforce.com

Phone: +1 604 783 1481
Email: pbryan@anode.ca

Mark Nottingham (editor)

Email: mnot@mnot.net