Internet Engineering Task Force (IETF)
Request for Comments: 6902
Category: Standards Track
ISSN: 2070-1721

P. Bryan, Ed.
Salesforce.com
M. Nottingham, Ed.
Akamai

April 2013

JavaScript Object Notation (JSON) Patch

Abstract

JSON Patch defines a JSON document structure for expressing a sequence of operations to apply to a JavaScript Object Notation (JSON) document; it is suitable for use with the HTTP PATCH method. The "application/json-patch+json" media type is used to identify such patch documents.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6902.

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+json") for expressing a sequence of operations to apply to a target JSON document; it is suitable for use with the HTTP PATCH method.

This format is also potentially useful in other cases in which it is necessary to make partial updates to a JSON document or to a data structure that has similar constraints (i.e., they can be serialized as an object or an array using the JSON grammar).

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].

3. Document Structure

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

The following is an example JSON Patch document, transferred in an HTTP PATCH request:

```
PATCH /my/data HTTP/1.1
Host: example.org
Content-Length: 326
Content-Type: application/json-patch+json
If-Match: "abc123"

[
  { "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", "from": "/a/b/c", "path": "/a/b/d" },
  { "op": "copy", "from": "/a/b/d", "path": "/a/b/e" }
]
```

Evaluation of a JSON Patch document begins against 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 until 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"; other values are errors. The semantics of each object is defined below.

Additionally, operation objects MUST have exactly one "path" member. That member's value is a string containing a JSON-Pointer value [RFC6901] that references a location within the target document (the "target location") where the operation is performed.

The meanings of other operation object members are defined by operation (see the subsections below). Members that are not explicitly defined for the operation in question MUST be ignored (i.e., the operation will complete as if the undefined member did not appear in the object).

Note that the ordering of members in JSON objects is not significant; therefore, the following operation objects 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" }
```

Operations are applied to the data structures represented by a JSON document, i.e., after any unescaping (see [RFC4627], Section 2.5) takes place.

4.1. 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.

The operation object MUST contain a "value" member whose content specifies the value to be added.

For example:

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

When the operation is applied, the target location 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 object - whereupon the supplied value is added to that object at the indicated location. If the member already exists, it is replaced by the specified value.

- An element to add to an existing array - whereupon the supplied value is added to the array at the indicated location. 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. If the "-" character is used to index the end of the array (see [RFC6901]), this has the effect of appending the value to the array.

Because this operation is designed to add to existing objects and arrays, its target location will often not exist. Although the pointer's error handling algorithm will thus be invoked, this specification defines the error handling behavior for "add" pointers to ignore that error and add the value as specified.

However, the object itself or an array containing it does need to exist, and it remains an error for that not to be the case. For example, an "add" with a target location of "/a/b" starting with this document:

```
{ "a": { "foo": 1 } }
```

is not an error, because "a" exists, and "b" will be added to its value. It is an error in this document:

```
{ "q": { "bar": 2 } }
```

because "a" does not exist.

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 whose content 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 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 a specified location and adds it to the target location.

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

The "from" location MUST exist for the operation to be successful.

For example:

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

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.

4.5. copy

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

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

The "from" location MUST exist for the operation to be successful.

For example:

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

This operation is functionally identical to an "add" operation at the target location using the value specified in the "from" member.

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 the target location’s value.

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 that they are considered equal by the following rules for that type:

- strings: are considered equal if they contain the same number of Unicode characters and their code points are byte-by-byte equal.
- numbers: are considered equal if their values are numerically equal.
- arrays: are considered equal if they contain the same number of values, and if each value can be considered equal to the value at the corresponding position in the other array, using this list of type-specific rules.

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

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

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

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

For example:

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

5. Error Handling

If a normative 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 the HTTP PATCH method is atomic, as per [RFC5789]. 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+json.

Type name: application

Subtype name: json-patch+json

Required parameters: none

Optional parameters: none

Encoding considerations: binary

Security considerations:
See Security Considerations in Section 7.

Interoperability considerations: N/A

Published specification:
RFC 6902

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 [RFC6901].

A few older Web browsers can be coerced into loading an arbitrary JSON document whose root is an array, leading to a situation in which a JSON Patch document containing sensitive information could be exposed to attackers, even if access is authenticated. This is known as a Cross-Site Request Forgery (CSRF) attack [CSRF].

However, such browsers are not widely used (at the time of writing, it is estimated that they are used in less than 1% of the market). Publishers who are nevertheless concerned about this attack are advised to avoid making such documents available with HTTP GET.

8. Acknowledgements

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

Mike Acar, Mike Amundsen, Cyrus Daboo, Paul Davis, Stefan Koegl, Murray S. Kucherawy, Dean Landolt, Randall Leeds, James Manger, Julian Reschke, James Snell, Eli Stevens, and Henry S. Thompson.

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

9. References

9.1. Normative References

[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.

[RFC6901] Bryan, P., Ed., Zyp, K., and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Pointer", RFC 6901, April 2013.

9.2. Informative References

[CSRF] Barth, A., Jackson, C., and J. Mitchell, "Robust Defenses for Cross-Site Request Forgery", ACM Conference on Computer and Communications Security, October 2008, <http://seclab.stanford.edu/websec/csrf/csrf.pdf>.

[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", "from": "/foo/waldo", "path": "/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", "from": "/foo/1", "path": "/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 Nonexistent 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, it would not be 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, because it contains a later "op":"remove" element. JSON requires that object member names be unique with a "SHOULD" requirement, and there is no standard error handling for duplicates.

A.14. ~ Escape Ordering

An example target JSON document:

```
{
  "/": 9,
  "~1": 10
}
```

A JSON Patch document:

```
[
  {"op": "test", "path": "/~01", "value": 10}
]
```

The resulting JSON document:

```
{
  "/": 9,
  "~1": 10
}
```

A.15. Comparing Strings and Numbers

An example target JSON document:

```
{
  "/": 9,
  "~1": 10
}
```

A JSON Patch document:

```
[
  {"op": "test", "path": "/~01", "value": "10"}
]
```

This results in an error, because the test fails. The document value is numeric, whereas the value being tested for is a string.

A.16. Adding an Array Value

An example target JSON document:

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

A JSON Patch document:

```
[
  { "op": "add", "path": "/foo/-", "value": ["abc", "def"] }
]
```

The resulting JSON document:

```
{ "foo": ["bar", ["abc", "def"]] }
```

Authors' Addresses

Paul C. Bryan (editor)
Salesforce.com

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

Mark Nottingham (editor)
Akamai

EMail: mnot@mnot.net