Applications Area Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 8, 2013

P. Bryan, Ed.
Salesforce.com
M. Nottingham, Ed.
Akamai
January 4, 2013

JSON Patch
draft-ietf-appsawg-json-patch-09

Abstract

JSON Patch defines the media type "application/json-patch", a JSON document structure for expressing a sequence of operations to apply to a JavaScript Object Notation (JSON) document, suitable for use with the HTTP PATCH method.

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 July 8, 2013.

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.

This format is also potentially useful in other cases where necessary to make partial updates to a JSON document.

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 that represents 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", "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 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 a location within the target document to perform the operation (the "target location").

The meanings of other members of operation objects are defined by the operation (see the subsections below). Members that are not explicitly defined for the operation in question MUST be ignored.

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 the following function, 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.

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 [JSON-Pointer]), 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 behaviour 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, "add"ing to the path "/a/b" to 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 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 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, 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, 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".

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 that conveyed by "value" are of the same JSON type, and 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 position-wise equal.

- numbers: are considered equal if their values are numerically equal.

- 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, using this list of type-specific rules.

- 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.

- 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 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.

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].

A few older Web browsers can be coerced into loading an arbitrary JSON document whose root is an array, leading to a situation where 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 (estimated to comprise less than 1% of the market, at the time of writing). 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 [RFC5261] specification.

9. References

9.1. Normative References

[JSON-Pointer] Bryan, P., Zyp, K., and M. Nottingham, "JSON Pointer", draft-ietf-appsawg-json-pointer-07 (work in progress), November 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

[CSRF] Barth, A., Jackson, C., and J. Mitchell, "Robust Defenses for Cross-Site Request Forgery".

[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 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).

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 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