Workgroup: Network Working Group
Internet-Draft:
draft-goessner-dispatch-jsonpath-00
Published: 12 July 2020
Intended Status: Standards Track
Expires: 13 January 2021
Authors: S. Gössner   C. Bormann, Ed.
         Fachhochschule Dortmund   Universität Bremen TZI

JSONPath -- XPath for JSON

Abstract

insert abstract here

Contributing

This document picks up the popular JSONPath specification dated 2007-02-21 and provides a more normative definition for it. It is intended as a submission to the IETF DISPATCH WG, in order to find the right way to complete standardization of this specification. In its current state, it is a strawman document showing what needs to be covered.

Comments and issues can be directed at the github repository *insert repo here* as well as (for the time when the more permanent home is being decided) at the dispatch@ietf.org mailing list.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document picks up the popular JSONPath specification dated 2007-02-21 [JSONPath-orig] and provides a more normative definition for it. It is intended as a submission to the IETF DISPATCH WG, in order to find the right way to complete standardization of this specification. In its current state, it is a strawman document showing what needs to be covered.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in

BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The grammatical rules in this document are to be interpreted as described in [RFC5234].

The terminology of [RFC8259] applies.

Data Item: A structure complying to the generic data model of JSON, i.e., composed of containers such as arrays and maps (JSON objects), and of atomic data such as null, true, false, numbers, and text strings.

Object: Used in its generic sense, e.g., for programming language objects. When a JSON Object as defined in [RFC8259] is meant, we specifically say JSON Object.

Query: Short name for JSONPath expression.

Argument: Short name for the JSON data item a JSONPath expression is applied to.

Output Path: A simple form of JSONPath expression that identifies a Position by providing a query that results in exactly that position. Similar to, but syntactically different from, a JSON Pointer [RFC6901].

Position: A JSON data item identical to or nested within the JSON data item to which the query is applied to, expressed either by the value of that data item or by providing a JSONPath Output Path.

1.2. Inspired by XPath

A frequently emphasized advantage of XML is the availability of plenty tools to analyse, transform and selectively extract data out of XML documents. [XPath] is one of these powerful tools.

In 2007, the need for something solving the same class of problems for the emerging JSON community became apparent, specifically for:

* Finding data interactively and extracting them out of [RFC8259] data items without special scripting.

* Specifying the relevant parts of the JSON data in a request by a client, so the server can reduce the data in the server response, minimizing bandwidth usage.

So how does such a tool look like when done for JSON? When defining a JSONPath, how should expressions look like?

The XPath expression

```
/store/book[1]/title
```

looks like

```
x.store.book[0].title
```

or

```
x['store']['book'][0]['title']
```

in popular programming languages such as JavaScript, Python and PHP, with a variable x holding the JSON data item. Here we observe that such languages already have a fundamentally XPath-like feature built in.

The JSONPath tool in question should:

*be naturally based on those language characteristics.

*cover only essential parts of XPath 1.0.

*be lightweight in code size and memory consumption.

*be runtime efficient.

1.3. Overview of JSONPath Expressions

JSONPath expressions always apply to a JSON data item in the same way as XPath expressions are used in combination with an XML document. Since a JSON data item is usually anonymous and doesn't necessarily have a "root member object", JSONPath used the abstract name $ to refer to the top level object of the data item.

JSONPath expressions can use the *dot-notation*

```
$.store.book[0].title
```

or the *bracket-notation*

```
$['store']['book'][0]['title']
```

for paths input to a JSONPath processor. Where a JSONPath processor uses JSONPath expressions for internal purposes or as output paths, these will always be converted to the more general *bracket-notation*.

JSONPath allows the wildcard symbol * for member names and array indices. It borrows the descendant operator .. from [E4X] and the array slice syntax proposal [start:end:step] [SLICE] from ECMASCRIPT 4.

JSONPath can employ an *underlying scripting language*, expressions of which, written in parentheses: (<expr>), can be used as an alternative to explicit names or indices as in:

```
$.store.book[(@.length-1)].title
```

The symbol @ is used for the current object. Filter expressions are supported via the syntax ?(<boolean expr>) as in

```
$.store.book[?(@.price < 10)].title
```

Here is a complete overview and a side by side comparison of the JSONPath syntax elements with its XPath counterparts.

XPath	JSONPath	Description
/	$	the root object/element
.	@	the current object/element
/	. or []	child operator
..	n/a	parent operator
//	..	nested descendants. JSONPath borrows this syntax from E4X.
*	*	wildcard. All objects/elements regardless of their names.
@	n/a	attribute access. JSON data items don't have attributes.
[]	[]	subscript operator. XPath uses it to iterate over element collections and for predicates. In JavaScript and JSON it is the native array operator.
|	[,]	Union operator in XPath results in a combination of node sets. JSONPath allows alternate names or array indices as a set.
n/a	[start:end:step]	array slice operator borrowed from ES4.
[]	?()	applies a filter (script) expression.
n/a	()	script expression, using the underlying script engine.
()	n/a	grouping in Xpath

Table 1: Overview over JSONPath, comparing to XPath

XPath has a lot more to offer (location paths in unabbreviated syntax, operators and functions) than listed here. Moreover there is a significant difference how the subscript operator works in Xpath and JSONPath:

* Square brackets in XPath expressions always operate on the *node set* resulting from the previous path fragment. Indices always start at 1.

* With JSONPath, square brackets operate on the *object* or *array* addressed by the previous path fragment. Indices always start at 0.

2. JSONPath Examples

This section provides some more examples for JSONPath expressions. The examples are based on a simple JSON data item patterned after a typical XML example representing a bookstore (that also has bicycles):

```
{ "store": {
    "book": [
      { "category": "reference",
        "author": "Nigel Rees",
        "title": "Sayings of the Century",
        "price": 8.95
      },
      { "category": "fiction",
        "author": "Evelyn Waugh",
        "title": "Sword of Honour",
        "price": 12.99
      },
      { "category": "fiction",
        "author": "Herman Melville",
        "title": "Moby Dick",
        "isbn": "0-553-21311-3",
        "price": 8.99
      },
      { "category": "fiction",
        "author": "J. R. R. Tolkien",
        "title": "The Lord of the Rings",
        "isbn": "0-395-19395-8",
        "price": 22.99
      }
    ],
    "bicycle": {
      "color": "red",
      "price": 19.95
    }
  }
}
```

Figure 1: Example JSON data item

The examples in Table 2 presume an underlying script language that allows obtaining the number of items in an array, testing for the presence of a map member, and performing numeric comparisons of map member values with a constant.

XPath	**JSONPath**	**Result**
/store/book/author	$.store.book[*].author	the authors of all books in the store
//author	$..author	all authors
/store/*	$.store.*	all things in store, which are some books and a red bicycle.
/store//price	$.store..price	the price of everything in the store.

XPath	JSONPath	Result
//book[3]	$..book[2]	the third book
//book[last()]	$..book[(@.length-1)] $..book[-1:]	the last book in order.
// book[position()<3]	$..book[0,1] $..book[:2]	the first two books
//book[isbn]	$..book[?(@.isbn)]	filter all books with isbn number
//book[price<10]	$..book[?(@.price<10)]	filter all books cheapier than 10
//*	$..*	all Elements in XML document. All members of JSON data item.

Table 2: Example JSONPath expressions applied to the example JSON data item

3. Detailed definition

[TBD: This section needs to be fleshed out in detail. The text given here is intended to give the flavor of that detail, not to be the actual definition that is to be defined.]

JSONPath expressions, "queries" for short in this specification, are character strings, represented in UTF-8 unless otherwise required by the context in which they are used.

When applied to a JSON data item, a query returns a (possibly empty) list of "positions" in the data item that match the JSONPath expression.

```
JSONPath = root *(step)
root = "$"

step = ".." name ; nested descendants
     / "." name ; child (dot notation)
     / "[" value-expression *("," value-expression) "]"
        ; child[ren] (bracket notation)
     / "[" value-expression *2(":" value-expression) "]"  ; (slice)
value-expression = *DIGIT / name
                 / script-expression / filter-expression
name = "'" text "'"
     / "*" ; wildcard
script-expression = "(" script ")"
filter-expression = "?(" script ")"
script = <To be defined in the course of standardization>
text = <any text, restrictions to be defined>
DIGIT = %x30-39
```

Figure 2: ABNF definition for JSONPath

Within a script, @ stands for the position under consideration. [TBD: define underlying scripting language, if there is to be a standard one]

[TBD: define handling of spaces]

A JSONPath starts at the root of the argument; the "current list" of positions is initialized to that root. Each step applies the semantics of that step to each of the positions in the "current list", returning another list; the "current list" is replaced by the concatenation of all these returned lists, and the next step begins. When all steps have been performed, the "current list" is returned, depending on the choices of the context either as a list of data items or as a list of output paths. [TBD: define the order of that list]

[TBD: Define all the steps]

[TBD: Define details of Output Path]

4. Discussion

* Currently only single quotes allowed inside of JSONPath expressions.

* Script expressions inside of JSONPath locations are currently not recursively evaluated by jsonPath. Only the global $ and local @ symbols are expanded by a simple regular expression.

* An alternative for jsonPath to return false in case of no match may be to return an empty array in future. [This is already done in the above.]

5. IANA considerations

TBD: Define a media type for JSON Path expressions.

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI

10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

6.2. Informative References

[E4X] ISO, "Information technology -- ECMAScript for XML (E4X) specification", ISO/IEC 22537:2006 , 2006.

[JSON-PHP] "JSON-PHP", January 2005, <http://mike.teczno.com/json.html>.

[JSONPath-impl] "jsonpath (Downloads)", n.d., <https://code.google.com/archive/p/jsonpath/downloads>.

[JSONPath-orig] Gössner, S., "JSONPath - XPath for JSON", 21 February 2007, <https://goessner.net/articles/JsonPath/>.

[RFC6901] Bryan, P., Ed., Zyp, K., and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Pointer", RFC 6901, DOI 10.17487/RFC6901, April 2013, <https://www.rfc-editor.org/info/rfc6901>.

[SLICE] "Slice notation", n.d., <https://github.com/tc39/proposal-slice-notation>.

[XPath] Berglund, A., Boag, S., Chamberlin, D., Fernandez, M., Kay, M., Robie, J., and J. Simeon, "XML Path Language (XPath) 2.0 (Second Edition)", World Wide Web Consortium Recommendation REC-xpath20-20101214, 14 December 2010, <http://www.w3.org/TR/2010/REC-xpath20-20101214>.

Appendix A. Early JSONPath implementations

This appendix has been copied from the similar section in [JSONPath-orig], with few changes. It is informative, intended to supply more examples and give an impression for what could be a typical JSONPath API.

A.1. Implementation

JSONPath is implemented in JavaScript for client-side usage and ported over to PHP for use on the server.

A.2. Usage

All you need to do is downloading either of the files

*jsonpath.js [JSONPath-impl]

*jsonpath.php [JSONPath-impl]

include it in your program and use the simple API consisting of one single function.

```
jsonPath(obj, expr [, args])
```

A.3. Parameters

obj (object|array): Object representing the JSON data item.

expr (string): JSONPath expression string.

args (object|undefined): Object controlling path evaluation and output. Currently only one member is supported.

args.resultType ("VALUE"|"PATH"): causes the result to be either matching values (default) or normalized path expressions.

A.4. Return value

(array|false): Array holding either values or normalized path expressions matching the input path expression, which can be used for lazy evaluation. false in case of no match.

A.5. JavaScript Example

```
var o = { /*...*/ },  // the 'store' JSON object from above
    res1 = jsonPath(o, "$..author").toJSONString(),
    res2 = jsonPath(o, "$..author",
                    {resultType:"PATH"}).toJSONString();
```

A.6. PHP example

We need here to convert the JSON string to a PHP array first. I am using Michal Migurski's JSON parser [JSON-PHP] for that.

```
require_once('json.php');      // JSON parser
require_once('jsonpath.php');  // JSONPath evaluator

$json = '{ ... }';  // JSON data item from above

$parser = new Services_JSON(SERVICES_JSON_LOOSE_TYPE);
$o = $parser->decode($json);
$match1 = jsonPath($o, "$..author");
$match2 = jsonPath($o, "$..author",
                   array("resultType" => "PATH"));
$res1 = $parser->encode($match1);
$res2 = $parser->encode($match2);
```

A.7. Results

Both JavaScript and PHP example result in the following JSON arrays (as strings):

```
res1:
[ "Nigel Rees",
  "Evelyn Waugh",
  "Herman Melville",
  "J. R. R. Tolkien"
]
res2:
[ "$['store']['book'][0]['author']",
  "$['store']['book'][1]['author']",
  "$['store']['book'][2]['author']",
  "$['store']['book'][3]['author']"
]
```

Please note that the return value of jsonPath is an array, which is also a valid JSON data item. So you might want to apply jsonPath to the resulting data item again or use one of your favorite array methods as sort with it.

Acknowledgements

This specification is based on Stefan Gössner's original online article defining JSONPath [JSONPath-orig].

The books example was taken from http://coli.lili.uni-bielefeld.de/~andreas/Seminare/sommer02/books.xml -- a dead link now.

Authors' Addresses

Stefan Gössner
Fachhochschule Dortmund
Sonnenstraße 96
D-44139 Dortmund

Germany

Email: stefan.goessner@fh-dortmund.de

Carsten Bormann (editor)
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org