Applications Area Working Group
Internet-Draft
Intended status: Informational
Expires: January 5, 2013

P. Bryan, Ed.
ForgeRock
K. Zyp
SitePen (USA)
M. Nottingham, Ed.
Rackspace

July 4, 2012

JSON Pointer
draft-ietf-appsawg-json-pointer-02

Abstract

JSON Pointer defines a string syntax for identifying a specific value within 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 January 5, 2013.

Table of Contents

1. Introduction

This specification defines JSON Pointer, a string syntax for identifying a specific value within a JavaScript Object Notation (JSON) [RFC4627] document. It is intended to be easily expressed in JSON string values and Uniform Resource Identifier (URI) [RFC3986] fragment identifiers.

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 [RFC2119].

This specification expresses normative syntax rules using Augmented Backus-Naur Form [RFC5234] (ABNF) notation.

3. Syntax

A JSON Pointer is a [Unicode] string containing a sequence of zero or more reference tokens, each prefixed by a '/' (%x2F) character.

If a reference token contains '~' (%x7E) or '/' (%x2F) characters, they MUST be encoded as '~0' and '~1' respectively.

ABNF syntax:

```
json-pointer = *( "/" reference-token )
reference-token = *( unescaped / escaped )
unescaped = %x00-2E / %x30-7D / %x7F-10FFFF
escaped = "~" ( "0" / "1" )
```

It is an error condition if a JSON Pointer value does not conform to this syntax (see Section 7).

4. Evaluation

Evaluation of a JSON Pointer begins with a reference to the root value of a JSON text document and completes with a reference to some value within the document. Each reference token in the JSON Pointer is sequentially evaluated.

Evaluation of each reference token begins by decoding any escaped character sequence; this is performed by first transforming any occurrence of the sequence '~1' to '/', then transforming any

occurrence of the sequence '~0' to '~'.

The reference token then modifies which value is referenced according to the following scheme:

> If the currently referenced value is a JSON object, the new referenced value is the object member with the name (after unescaping any backslash escape sequences that can occur in a JSON string) identified by the reference token. The member name is equal to the token if it has the same number of Unicode characters as token and their code points are position-wise equal. If a referenced member name is not unique in an object, the member that is referenced is undefined.
>
> If the currently referenced value is a JSON array, the reference token MUST contain characters that represent an unsigned base-10 integer value (possibly with leading zeros), and the new referenced value is the array element with the zero-based index identified by the token.

If a reference token is being evaluated against a JSON document, the implementation MAY evaluate each token against a concrete value, and terminate evaluation with an error condition if a evaluation fails to resolve a concrete value (see Section 7).

5. JSON String Representation

A JSON Pointer can be represented in a JSON string value. Per [RFC4627], section 2.5, all instances of quotation mark '"' (%x22), reverse solidus '\' (%x5C) and control (%x00-1F) characters MUST be escaped.

For example, given the JSON document

```
{
   "foo": ["bar", "baz"],
   "": 0
   "a/b": 1,
   "c%d": 2,
   "e^f": 3,
   "g|h": 4,
   "i\\j": 5,
   "k\"l": 6,
   " ": 7,
   "m~n": 8
}
```

Then the following JSON strings evaluate to the accompanying values:

```
 ""         // the whole document
 "/foo"       ["bar", "baz"]
 "/foo/0"    "bar"
 "/"          0
 "/a~1b"      1
 "/c%d"       2
 "/e^f"       3
 "/g|h"       4
 "/i\\j"      5
 "/k\"l"      6
 " "          7
 "m~0n"       8
```

6. URI Fragment Identifier Representation

A JSON Pointer can be represented in a URI fragment identifier. by encoding it into octets, using UTF-8 [RFC3629], percent-encoding those characters not allowed by the fragment rule in [RFC3986].

Given the same example document as above, the following URI fragment identifiers evaluate to the accompanying values:

```
 #                  // the whole document
 #/foo            ["bar", "baz"]
 #/foo/0         "bar"
 #/               0
 #/a~1b           1
 #/c%25d          2
 #/e%5Ef          3
 #/g%7Ch          4
 #/i%5Cj          5
 #/k%22l          6
 #/%20            7
 #/m~0n           8
```

7. Error Handling

In the event of an error condition, evaluation of the JSON Pointer fails to complete.

This includes, but is not limited to:

- Invalid pointer syntax
- A pointer that references a non-existent value

This specification does not define how errors are handled; an application of JSON Pointer SHOULD specify the impact and handling of each type of error.

For example, some applications might stop pointer processing upon an error; others may attempt to recover from missing values by inserting default ones.

8. IANA Considerations

TBD

9. Security Considerations

A given JSON Pointer is not guaranteed to reference an actual JSON value. Implementations should be aware of this and take appropriate precautions.

Note that JSON pointers can contain the NUL (Unicode U+0000) character, which may not be representable in all programming languages.

10. Acknowledgements

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

Mike Acar, Carsten Bormann, Tim Bray, Jacob Davies, Martin J. Duerst, Bjoern Hoehrmann, James H. Manger, Drew Perttula, Julian Reschke.

11. Normative References

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

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform

Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

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

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[Unicode] The Unicode Consortium, "The Unicode Standard, Version 6.0", October 2011, <http://www.unicode.org/versions/Unicode6.0.0/>.

Authors' Addresses

Paul C. Bryan (editor)
ForgeRock

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

Kris Zyp
SitePen (USA)

Phone: +1 650 968 8787
Email: kris@sitepen.com

Mark Nottingham (editor)
Rackspace

Email: mnot@mnot.net