                   JavaScript Object Notation (JSON) Pointer

Abstract

   JSON Pointer defines a string syntax for identifying a specific value
   within a JavaScript Object Notation (JSON) document.

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/rfc6901.

Table of Contents

1. Introduction

   This specification defines JSON Pointer, a string syntax for
   identifying a specific value within a JavaScript Object Notation
   (JSON) document [RFC4627]. JSON Pointer is intended to be easily
   expressed in JSON string values as well as 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 (ABNF) [RFC5234] notation.

3. Syntax

   A JSON Pointer is a Unicode string (see [RFC4627], Section 3)
   containing a sequence of zero or more reference tokens, each prefixed
   by a '/' (%x2F) character.

   Because the characters '~' (%x7E) and '/' (%x2F) have special
   meanings in JSON Pointer, '~' needs to be encoded as '~0' and '/'
   needs to be encoded as '~1' when these characters appear in a
   reference token.

The ABNF syntax of a JSON Pointer is:

```
json-pointer    = *( "/" reference-token )
reference-token = *( unescaped / escaped )
unescaped       = %x00-2E / %x30-7D / %x7F-10FFFF
   ; %x2F ('/') and %x7E ('~') are excluded from 'unescaped'
escaped         = "~" ( "0" / "1" )
   ; representing '~' and '/', respectively
```

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

Note that JSON Pointers are specified in characters, not as bytes.

4. Evaluation

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

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 '/', and then transforming any
occurrence of the sequence '~0' to '~'. By performing the
substitutions in this order, an implementation avoids the error of
turning '~01' first into '~1' and then into '/', which would be
incorrect (the string '~01' correctly becomes '~1' after
transformation).

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

o If the currently referenced value is a JSON object, the new
  referenced value is the object member with the name identified by
  the reference token. The member name is equal to the token if it
  has the same number of Unicode characters as the token and their
  code points are byte-by-byte equal. No Unicode character
  normalization is performed. If a referenced member name is not
  unique in an object, the member that is referenced is undefined,
  and evaluation fails (see below).

   o If the currently referenced value is a JSON array, the reference
      token MUST contain either:

      * characters comprised of digits (see ABNF below; note that
         leading zeros are not allowed) that represent an unsigned
         base-10 integer value, making the new referenced value the
         array element with the zero-based index identified by the
         token, or

      * exactly the single character "-", making the new referenced
         value the (nonexistent) member after the last array element.

   The ABNF syntax for array indices is:

   array-index = %x30 / (%x31-39 *(%x30-39))
                 ; "0", or digits without a leading "0"

   Implementations will evaluate each reference token against the
   document's contents and will raise an error condition if it fails to
   resolve a concrete value for any of the JSON pointer's reference
   tokens. For example, if an array is referenced with a non-numeric
   token, an error condition will be raised. See Section 7 for details.

   Note that the use of the "-" character to index an array will always
   result in such an error condition because by definition it refers to
   a nonexistent array element. Thus, applications of JSON Pointer need
   to specify how that character is to be handled, if it is to be
   useful.

   Any error condition for which a specific action is not defined by the
   JSON Pointer application results in termination of evaluation.

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.

   Note that before processing a JSON string as a JSON Pointer,
   backslash escape sequences must be unescaped.

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
}
```

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], while percent-encoding
   those characters not allowed by the fragment rule in [RFC3986].

   Note that a given media type needs to specify JSON Pointer as its
   fragment identifier syntax explicitly (usually, in its registration
   [RFC6838]). That is, just because a document is JSON does not imply
   that JSON Pointer can be used as its fragment identifier syntax. In
   particular, the fragment identifier syntax for application/json is
   not JSON Pointer.

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.

   Error conditions include, but are not limited to:

   o Invalid pointer syntax

   o A pointer that references a nonexistent 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, while others may attempt to recover from missing values by
   inserting default ones.

8. Security Considerations

   A given JSON Pointer is not guaranteed to reference an actual JSON
   value. Therefore, applications using JSON Pointer should anticipate
   this situation by defining how a pointer that does not resolve ought
   to be handled.

   Note that JSON pointers can contain the NUL (Unicode U+0000)
   character. Care is needed not to misinterpret this character in
   programming languages that use NUL to mark the end of a string.

9. 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, and
      Julian Reschke.

10. References

10.1. 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.

10.2. Informative References

   [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type
              Specifications and Registration Procedures", BCP 13,
              RFC 6838, January 2013.

Authors' Addresses

    Paul C. Bryan (editor)
    Salesforce.com

    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)
    Akamai

    EMail: mnot@mnot.net