Internet Engineering Task Force (IETF)
Request for Comments: 7464
Category: Standards Track
ISSN: 2070-1721

N. Williams
Cryptonector

February 2015

JavaScript Object Notation (JSON) Text Sequences

Abstract

This document describes the JavaScript Object Notation (JSON) text sequence format and associated media type "application/json-seq". A JSON text sequence consists of any number of JSON texts, all encoded in UTF-8, each prefixed by an ASCII Record Separator (0x1E), and each ending with an ASCII Line Feed character (0x0A).

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/rfc7464.

Table of Contents

1. Introduction and Motivation

The JavaScript Object Notation (JSON) [RFC7159] is a very handy serialization format. However, when serializing a large sequence of values as an array, or a possibly indeterminate-length or never-ending sequence of values, JSON becomes difficult to work with.

Consider a sequence of one million values, each possibly one kilobyte when encoded -- roughly one gigabyte. It is often desirable to process such a dataset in an incremental manner without having to first read all of it before beginning to produce results. Traditionally, the way to do this with JSON is to use a "streaming" parser, but these are not widely available, widely used, or easy to use.

This document describes the concept and format of "JSON text sequences", which are specifically not JSON texts themselves but are composed of (possible) JSON texts. JSON text sequences can be parsed (and produced) incrementally without having to have a streaming parser (nor streaming encoder).

1.1. Conventions Used in This Document

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 [RFC2119].

2. JSON Text Sequence Format

Two different sets of ABNF rules are provided for the definition of JSON text sequences: one for parsers and one for encoders. Having two different sets of rules permits recovery by parsers from sequences where some of the elements are truncated for whatever reason. The syntax for parsers is specified in terms of octet strings that are then interpreted as JSON texts, if possible. The syntax for encoders, on the other hand, assumes that sequence elements are not truncated.

JSON text sequences MUST use UTF-8 encoding; other encodings of JSON (i.e., UTF-16 and UTF-32) MUST NOT be used.

2.1. JSON Text Sequence Parsing

The ABNF [RFC5234] for the JSON text sequence parser is as given in Figure 1.

```
input-JSON-sequence = *(1*RS possible-JSON)
RS = %x1E; "record separator" (RS), see RFC 20
          ; Also known as: Unicode Character INFORMATION SEPARATOR
          ;                TWO (U+001E)
possible-JSON = 1*(not-RS); attempt to parse as UTF-8-encoded
                          ; JSON text (see RFC 7159)
not-RS = %x00-1d / %x1f-ff; any octets other than RS
```

Figure 1: JSON Text Sequence ABNF

In prose: a series of octet strings, each containing any octet other than a record separator (RS) (0x1E) [RFC20]. All octet strings are preceded by an RS byte. Each octet string in the sequence is to be parsed as a JSON text in the UTF-8 encoding [RFC3629].

If parsing of such an octet string as a UTF-8-encoded JSON text fails, the parser SHOULD nonetheless continue parsing the remainder of the sequence. The parser can report such failures to applications, which might then choose to terminate parsing of a sequence. Multiple consecutive RS octets do not denote empty sequence elements between them and can be ignored.

This document does not define a mechanism for reliably identifying text sequence by position (for example, when sending individual elements of an array as unique text sequences). For applications where truncation is a possibility, this means that intended sequence elements can be truncated and can even be missing entirely; therefore, a reference to an nth element would be unreliable.

There is no end of sequence indicator.

2.2. JSON Text Sequence Encoding

The ABNF for the JSON text sequence encoder is given in Figure 2.

```
JSON-sequence = *(RS JSON-text LF)
RS = %x1E; see RFC 20
          ; Also known as: Unicode Character INFORMATION SEPARATOR
          ;                TWO (U+001E)
LF = %x0A; "line feed" (LF), see RFC 20
JSON-text = <given by RFC 7159, using UTF-8 encoding>
```

Figure 2: JSON Text Sequence ABNF

In prose: any number of JSON texts, each encoded in UTF-8 [RFC3629], each preceded by one ASCII RS character, and each followed by a line feed (LF). Since RS is an ASCII control character, it may only appear in JSON strings in escaped form (see [RFC7159]), and since RS may not appear in JSON texts in any other form, RS unambiguously delimits the start of any element in the sequence. RS is sufficient to unambiguously delimit all top-level JSON value types other than numbers. Following each JSON text in the sequence with an LF allows detection of truncated JSON texts consisting of a number at the top-level; see Section 2.4.

JSON text sequence encoders are expected to ensure that the sequence elements are properly formed. When the JSON text sequence encoder does the JSON text encoding, the sequence elements will naturally be properly formed. When the JSON text sequence encoder accepts already-encoded JSON texts, the JSON text sequence encoder ought to parse them before adding them to a sequence.

Note that on some systems it"s possible to input RS by typing "ctrl-^"; on some system or applications, the correct sequence may be "ctrl-v ctrl-^". This is helpful when constructing a sequence manually with a text editor.

2.3. Incomplete/Invalid JSON Texts Need Not Be Fatal

Per Section 2.1, JSON text sequence parsers should not abort when an octet string contains a malformed JSON text. Instead, the JSON text sequence parser should skip to the next RS. Such a situation may arise in contexts where, for example, data that is appended to log files to log files is truncated by the filesystem (e.g., due to a crash or administrative process termination).

Incremental JSON text parsers may be used, though of course failure to parse a given text may result after first producing some incremental parse results.

Sequence parsers should have an option to warn about truncated JSON texts.

2.4. Top-Level Values: numbers, true, false, and null

While objects, arrays, and strings are self-delimited in JSON texts, numbers and the values 'true', 'false', and 'null' are not. Only whitespace can delimit the latter four kinds of values.

JSON text sequences use 0x0A as a "canary" octet to detect truncation.

Parsers MUST check that any JSON texts that are a top-level number, or that might be 'true', 'false', or 'null', include JSON whitespace (at least one byte matching the "ws" ABNF rule from [RFC7159]) after that value; otherwise, the JSON-text may have been truncated. Note that the LF following each JSON text matches the "ws" ABNF rule.

Parsers MUST drop JSON-text sequence elements consisting of non-self-delimited top-level values that may have been truncated (that are not delimited by whitespace). Parsers can report such texts as warnings (including, optionally, the parsed text and/or the original octet string).

For example, '<RS>123<RS>' might have been intended to carry the top-level number 1234, but it got truncated. Similarly, '<RS>true<RS>' might have been intended to carry the invalid text 'trueish'. '<RS>truefalse<RS>' is not two top-level values, 'true', and 'false'; it is simply not a valid JSON text.

Implementations may produce a value when parsing '<RS>"foo"<RS>' because their JSON text parser might be able to consume bytes incrementally; since the JSON text in this case is a self-delimiting top-level value, the parser can produce the result without consuming an additional byte. Such implementations ought to skip to the next RS byte, possibly reporting any intervening non-whitespace bytes.

Detection of truncation of non-self-delimited sequence elements (numbers, true, false, and null) is only possible when the sequence encoder produces or receives complete JSON texts. Implementations where the sequence encoder is not also in charge of encoding the individual JSON texts should ensure that those JSON texts are complete.

3. Security Considerations

All the security considerations of JSON [RFC7159] apply. This format provides no cryptographic integrity protection of any kind.

As usual, parsers must operate on input that is assumed to be untrusted. This means that parsers must fail gracefully in the face of malicious inputs.

Note that incremental JSON text parsers can produce partial results and later indicate failure to parse the remainder of a text. A sequence parser that uses an incremental JSON text parser might treat a sequence like '<RS>"foo"<LF>456<LF><RS>' as a sequence of one element ("foo"), while a sequence parser that uses a non-incremental JSON text parser might treat the same sequence as being empty. This effect, and texts that fail to parse and are ignored, can be used to smuggle data past sequence parsers that don't warn about JSON text failures.

Repeated parsing and re-encoding of a JSON text sequence can result in the addition (or stripping) of trailing LF bytes from (to) individual sequence element JSON texts. This can break signature validation. JSON has no canonical form for JSON texts, therefore neither does the JSON text sequence format.

4. IANA Considerations

The MIME media type for JSON text sequences is application/json-seq.

Type name: application

Subtype name: json-seq

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See RFC 7464, Section 3.

Interoperability considerations: Described herein.

Published specification: RFC 7464.

Applications that use this media type:

<https://stedolan.github.io/jq>
<https://github.com/mapbox/cligj>
<https://github.com/hildjj/json-text-sequence>

Fragment identifier considerations: N/A

Additional information:

- Deprecated alias names for this type: N/A
- Magic number(s): N/A
- File extension(s): N/A
- Macintosh file type code(s): N/A

Person & email address to contact for further information:

json@ietf.org

Intended usage: COMMON

Author: Nicolas Williams (nico@cryptonector.com)

Change controller: IETF

5. Normative References

[RFC20] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, October 1969, <http://www.rfc-editor.org/info/rfc20>.

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

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003, <http://www.rfc-editor.org/info/rfc3629>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

Acknowledgements

Phillip Hallam-Baker proposed the use of JSON text sequences for logfiles and pointed out the need for resynchronization. Stephen Dolan created <https://github.com/stedolan/jq>, which uses something like JSON text sequences (with LF as the separator between texts on output, and requiring only such whitespace as needed to disambiguate on input). Carsten Bormann suggested the use of ASCII RS, and Joe Hildebrand suggested the use of LF in addition to RS for disambiguating top-level number values. Paul Hoffman shepherded the document. Many others contributed reviews and comments on the JSON Working Group mailing list.

Author's Address

Nicolas Williams
Cryptonector, LLC

EMail: nico@cryptonector.com