json   N. Williams
Internet-Draft   Cryptonector
Intended status: Standards Track   December 9, 2014
Expires: June 12, 2015

JavaScript Object Notation (JSON) Text Sequences
draft-ietf-json-text-sequence-10

Abstract

This document describes the JSON text sequence format and associated media type, "application/json-seq". A JSON text sequence consists of any number of JSON texts, each prefix by an Record Separator (U+001E), and each ending with a newline character (U+000A).

Status of this Memo

This Internet-Draft will expire on June 12, 2015.

Copyright Notice

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 1 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 neither widely available, widely used, nor 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 the elements are truncated for whatever reason. The syntax for parsers is specified in terms of octet strings which are then interpreted as JSON texts if possible. The syntax for encoders, on the other hand, assumes that sequence elements are not truncated.

2.1. JSON text sequence parsing

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

```
  JSON-sequence = *(1*RS possible-JSON)
  RS = %x1E; "record separator" (RS), see RFC20
  possible-JSON = 1*(not-RS); attempt to parse as UTF-8-encoded
                            ; JSON text (see RFC7159)
  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) [RFC0020], all octet strings separated from each other by RS octets. 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.

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 RFC20
         ; Also known as: Unicode Character 'INFORMATION SEPARATOR
         ;                TWO' (U+001E)
LF = %x0A; "line feed" (LF), see RFC20
JSON-text = <given by RFC7159, 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.

Note that on some systems it's possible to input RS by typing 'ctrl-^'. This is helpful when constructing a sequence manually with a text editor.

2.3. Incomplete JSON texts are not fatal

Per- Section 2.1, JSON text sequence parsers SHOULD NOT abort when an octet string contains a malformed JSON textm instead the JSON text sequence parser should skip to the next RS. Such a situation may arise in contexts where, for example, append-writes to log files are 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 numeric, 'true', 'false', and 'null' values

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.

Parsers MUST check that any JSON texts that are a top-level number, or which 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 123.4, but must have been 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, and 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.

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 as-good-as untrusted input. This means that parsers must fail gracefully in the face of malicious inputs. Note that incremental parsers can produce partial results and later indicate failure to parse the remainder of a text. Note that 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.

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 <this document, once published>, Section 3.

Interoperability considerations: Described herein.

Published specification: <this document, once published>.

Applications that use this media type: <by publication time <https://stedolan.github.io/jq> is likely to support this format>.

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: See the "Authors' Addresses" section of this document.
- Change controller: IETF

5. 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 Internet-Draft. Many others contributed reviews and comments on the JSON Working Group mailing list.

6. Normative References

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

[RFC0020] Cerf, V., "ASCII format for network interchange", RFC 20, October 1969.

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

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

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

Author's Address

Nicolas Williams
Cryptonector, LLC

Email: nico@cryptonector.com