json
Internet-Draft
Intended status: Standards Track
Expires: November 24, 2014

N. Williams
Cryptonector

May 23, 2014

JavaScript Object Notation (JSON) Text Sequences
draft-ietf-json-text-sequence-04

Abstract

This document describes the JSON text sequence format and associated media type.

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 November 24, 2014.

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, which would be 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 (see Section 1.1), 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 JSON texts. JSON text sequences can be parsed (and produced) incrementally without having to have a streaming parser (nor encoder).

1.1. JSON Parser Types

For the purposes of this document we shall classify JSON parsers as follows:

Streaming Consumes a text incrementally, outputs values incrementally (e.g., as (path, leaf value) pairs).

Online Consumes a text incrementally.

Off-line Consumes only complete texts.

1.2. Conventions used in this document

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].

2. JSON Text Sequence Format

The ABNF [RFC5234] for the JSON text sequence format is as given in Figure 1. Note that this ABNF does not work if we assume greedy matching. Therefore, in prose, a JSON text sequence is a sequence of zero or more JSON texts, each surrounded by any number of JSON whitespace characters and always followed by a newline.

```
JSON-sequence = ws *(JSON-text ws LF ws)
LF = <given by RFC5234>
ws = <given by RFC7159>
JSON-text = <given by RFC7159>
```

Figure 1: JSON text sequence ABNF

As long as a JSON text sequence consist of complete JSON texts, the only requirement is that whitespace separate any non-object, array, string top-level values from neighboring texts. The simplest way to ensure this is to require such whitespace, and furthermore it is convenient to use a newline, as we'll see in Section 2.1. Therefore we impose one requirement:

- JSON text sequence encoders MUST emit a newline after any JSON text.

2.1. Ambiguities

Otherwise An input of 'truefalse' is not a valid sequence of two JSON values, true and false! Neither is 'true0' a valid sequence of true and zero. Some existing JSON parsers that might be used to construct sequence parsers might in fact accept such sequences, resulting in erroneous parsing of sequences of two or more numbers. E.g., a sequence of two numbers, 4 and 2, encoded without the required whitespace between them would parse incorrectly as the number 42.

Such ambiguities is resolved by requiring that encoders emit a whitespace separator (specifically: a newline) after each text.

2.1.1. Ambiguities Resulting from Partial Texts

Another kind of ambiguity arises when a JSON text sequence contains partial texts. Such a sequence can result when using "append writes" to write to a file. For example, many systems might commit partial writes to stable storage then fail to complete the remainder of a write as a result of, e.g., power failures; upon recovery the file may then end with a partial JSON text.

[[anchor1: Perhaps we should add a note about what POSIX requires

w.r.t. O_APPEND, and how POSIX is agnostic as to power failures and so on. The point being that even where a standard imposes strong atomicity requirements as to append writes, there are good reasons why that might be difficult to obtain under exceptional circumstances.]]

Consider a portion of a JSON text sequence such as:

```
{ "foo":
{ "bar": 42 }
}
```

How can we tell that the first line isn't part of an incomplete JSON text? We can't, especially if the third line were missing.

In the common case JSON text sequence parsers assume every text is complete, and abort processing if any one text fails to parse. However, for logfiles, there is value is being able to recover from such situations. Recovery is described in Section 3.

2.2. Rationale for Choice of LF as the Text Separator

A variety of characters or character sequences (even non-whitespace characters) could have been used as the JSON text separator in JSON text sequences. The rationale for using newline (LF) as the separator is as follows:

- it matches the 'ws' ABNF rule in [RFC7159] (as do CR, HTAB, and SP);
- it is always escaped in encoded JSON strings, therefore it is safe remove LFs (or replace then with other JSON whitespace characters) from any JSON text (this is also true of CR and HTAB, but not SP);
- it is generally understood as the end-of-line marker by line-oriented tools;
- at least one JSON text sequence implementation exists and has existed for some time [XXX add external informative reference to https://stedolan.github.com/jq], and it uses LF as the JSON text separator.

Note that JSON text sequence writers may (and should) use CR LF as the text separator where the end-of-line marker is expected to be CR LF.

3. Use for Logfiles, or How to Resynchronize Following Truncated entries

The JSON Text Sequence format is useful for logfiles, as those are generally (and atomically) appended to on an ongoing basis. I.e., logfiles are of indeterminate length, at least right up until they are closed.

The partial-write ambiguities described in Section 2.1.1 come up in the case of logfiles.

As long as all texts in the logfile sequence are followed by a newline, it is possible to detect a subsequent JSON text written after an entry that fails to parse: either the first or the second subsequent, complete JSON texts. Figure 2 shows an ABNF rule for detecting the boundary between a non-truncated [and some truncated] JSON text and the next JSON text in a sequence. This rule assumes that only valid JSON texts are written to a sequence.

```
boundary = endchar *text-sep *ws startchar
text-sep = *(SP / HTAB / CR) LF ; these are from RFC5234
endchar = ( "}" / "]" / DQUOTE / "e" / "l" / DIGIT )
startchar =  ( "{" / "[" / DQUOTE / "t" / "f" / "n" / "-" / DIGIT )
ws = <given by RFC7159>
```

Figure 2: ABNF for resynchronization

To resynchronize after failing to parse a JSON text, simply search for a boundary as described in figure 2. A boundary found this way might be the boundary between the truncated entry and the subsequent entry, or it might be a subsequent boundary.

This method does not support scanning backwards for boundaries.

To make resynchronization reliable, and work both forwards and backwards, the writer MUST first ensure that the JSON text being written is valid, and SHOULD apply either (or both) of the following:

1. Remove internal newlines (not including escaped newlines in strings) from any JSON text being written.

2. Prefix any JSON text with a null value and a newline. The append write must still be atomic (one write), and contain both texts.

Method #1 permits scanning for newlines (in either direction) as the resynchronization method.

Method #2 permits scanning for "null" LF (in either direction) as the

resynchronization method.

Consider a JSON text sequence such as:

```
null
{ "foo":"hello world" }
"a broken writenull
"a complete write"
```

Resynchronization methods #1 and #2 will correctly detect that the third line is an incomplete JSON text, and that the next complete text starts at the fourth line. We can't tell which of method #1 or #2 the writer was using, but either method works for the parser. The parser SHOULD know which method the writer was using, as to know whether to discard the nulls, and whether to attempt resynchronization at all.

Method #1 is RECOMMENDED for JSON text sequence logfile writers.

4. Security Considerations

All the security considerations of JSON [RFC7159] apply.

There is no end of sequence indicator. This means that "end of file", "end of transmission", and so on, can be indistinguishable from a logical end of sequence. Applications where this matters should denote end of sequence by convention (e.g., Content-Length in HTTP).

The resynchronization ABNF heuristic is imperfect and might skip a valid entry following a truncated one. Purposefully appending a truncated (or invalid) JSON text to a JSON text sequence logfile can cause the subsequent entry to be invisible.

JSON text sequence writers MUST validate (parse) any JSON text inputs from untrusted third parties.

JSON text sequence logfile writers SHOULD apply one of the resynchronization methods described in Figure 2, preferably method #1.

5. 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 4.

Interoperability considerations: Described herein.

Published specification: <this document, once published>.

Applicat<http://xml2rfc.tools.ietf.org/public/rfc/bibxml/reference.RFC.2119.xml>ions that use this media type: JSON text sequences have been used in applications written with the jq programming language.

6. Acknowledgements

Phillip Hallam-Baker proposed the use of JSON text sequences for logfiles and pointed out the need for resynchronization. James Manger contributed the ABNF for resynchronization.

7. Normative References

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

[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