draft-jonas-json-template-language-01

tools.ietf.org/html/draft-jonas-json-template-language-01

Versions: 00 01

Network Working Group S. Jonas
Internet-Draft ETC Labs Core
Intended status: Informational March 29, 2019
Expires: September 30, 2019

JSON Template Language

Abstract

JSON Template Language is a simple, compact template language to substitute JSON [1] into a template. It is derived from the template literals of Javascript, as defined in the ECMAScript Programming Language Standard, Ninth Edition ECMA-262 [2]. It has uses not limited to: specifying interfaces, runtime expression mapping, and other forms of programmatically interacting with JSON and templates.

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 https://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 September 30, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 2
2. Conventions and Definitions 2
3. Security Considerations 3
4. IANA Considerations 3
5. Example Usage 3
5.1. Accessing Array indicies 3
5.2. Accessing Nested Objects and Arrays 4
6. JSON Template Grammar 4
6.1. Template literal 5
6.2. Identifier 5
6.3. Path 5
6.4. Array Index 5
6.5. Head 6
6.6. Tail 6
7. References 6
7.1. Normative References 6
7.2. URIs 6
Author's Address 7

1. Introduction

JSON Template Language provides a mechanism for describing various parts of a template by using familiar syntax from Javascript and JSON. By using a familiar syntax we can get interoperability within the JSON [3] and URI [4] specifications.

The terms "string", "object" and "array" are from the conventions of Javascript and JSON.

A JSON Template provides a structural description of a template, and when JSON is provided, machine-readable instructions on how to construct a string corresponding to those values. A template is transformed into a string by replacing each of the template literals with the corresponding JSON values.

2. Conventions and Definitions

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Security Considerations

When considering runtime security of JSON Template Language, the surface area is only expanding JSON value by key. No loops or operator grammar is defined and should be avoided in runtime contexts.

4. IANA Considerations

This document has no IANA actions.

5. Example Usage

for the given JSON:

```
{
  "query": {
    "number": 1,
    "salad": "potato"
  }
}
```

and the given template:

```
number${query.number}salad${query.salad}
       \_________/      \_________/
           |                |
           |                |
```

For each key in given JSON, substitute it by key into the string.

expansion:

```
number1saladpotato
```

5.1. Accessing Array indicies

for the given JSON

```
{
  "query": {
    "numbers": [0, 1, 2, 3],
    "salads": ["caesar", "potato"]
  }
}
```

and the given template:

```
number${query.numbers[1]}salad${query.salads[1]}
      \_____________/       \_____________/
            |                     |
            |                     |
```

For each key in given JSON, substitute it by array index into the string.

expansion:

number1saladpotato

5.2. Accessing Nested Objects and Arrays

for the given JSON

```
{
  "nested": {
    "query": {
      "number": 1,
      "salads": ["caesar", "potato"]
    }
  }
}
```

and the given template:

```
number${query.numbers[1]}salad${query.salads[1]}
      \_____________/       \_____________/
            |                     |
            |                     |
```

For each key in given JSON, substitute it by path first. then array index into the string.

expansion:

number1saladpotato

6. JSON Template Grammar

Language Grammar defined in ABNF [5]

grammar = *([head] template-head identifier *["." path] *[array-left array-index array-right] template-tail [tail])
template-head = "${"
template-tail = "}"
identifier = *(ALPHA / "_")
path = *(ALPHA / "_")
array-left = "["
array-right = "]"
array-index = *(DIGIT)
head = *(ALPHA / DIGIT / special-characters)
tail = *(ALPHA / DIGIT / special-characters)
special-characters = *("-" / "_" / "~" / "." / ":" / "/" / "?" / "#" / "[" / "]" / "@" / "!" / "&" / "'" / "(" / ")" / "*" / "+" / "," / ";" / "=")

6.1. Template literal

The contents within the Template literal should be valid a key-value object mapping or array-index mapping for JSON [6].

grammar = *([head] template-head identifier *["." path] \
[array-left array-index array-right] template-tail [tail])

A template literal is represented within a URIs [7] as a pair of curly braces starting with a "$".

template-head = "${"
template-tail = "}"

6.2. Identifier

An identifier is a valid JSON key.

identifier = *(ALPHA / "_")

6.3. Path

MUST be a valid nested JSON key that will be resolved with the identifier.

*["." path]

6.4. Array Index

MUST be a valid index of a JSON array are represented within square brackets and MUST be a valid number.

*[array-left array-index array-right]
array-left = "["
array-right = "]"
array-index = *(DIGIT)

6.5. Head

Head represents the characters before template-head, this could be beginning of the template or the last tail.

head = *(ALPHA / DIGIT / special-characters)

6.6. Tail

Tail represents the characters after the template-tail until the end of the template or the next instance of a template literal.

tail = *(ALPHA / DIGIT / special-characters)

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

7.2. URIs

[1] https://tools.ietf.org/html/rfc7159

[2] https://www.ecma-international.org/ecma-262/9.0/index.html#sec-template-literals

[3] https://tools.ietf.org/html/rfc7159, https://www.ecma-international.org/ecma-262/9.0/index.html#sec-template-literals

[4] https://tools.ietf.org/html/rfc6570

[5] https://tools.ietf.org/html/rfc5234

[6] https://tools.ietf.org/html/rfc7159

[7] https://tools.ietf.org/html/rfc3986

Author's Address

Shane Jonas
ETC Labs Core

Email: shane.j@etclabs.org

Html markup produced by rfcmarkup 1.129d, available from https://tools.ietf.org/tools/rfcmarkup/