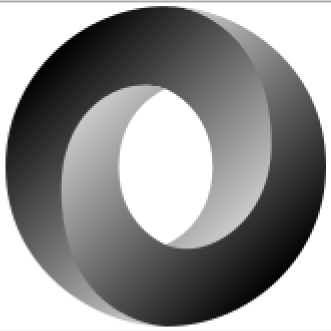
WIKIPEDIA

JSON

JavaScript Object Notation (**JSON**, pronounced /ˈdʒeɪsən/; also /ˈdʒeɪˌsɒn/[note 1]) is an open standard file format, and data interchange format, that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and array data types (or any other serializable value). It is a very common data format, with a diverse range of applications, such as serving as a replacement for XML in AJAX systems.[6]



Filename extension	`.json`
Internet media type	`application/json`
Type code	TEXT
Type of format	Data interchange
Extended from	JavaScript
Standard	STD 90 (https://tools.ietf.org/html/std90) (RFC 8259 (https://tools.ietf.org/html/rfc8259)), ECMA-404 (http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf), ISO/IEC 21778:2017 (https://www.iso.org/standard/71616.html)
Open format?	Yes
Website	json.org (https://json.org/)

JSON is a language-independent data format. It was derived from JavaScript, but many modern programming languages include code to generate and parse JSON-format data. The official Internet media type for JSON is `application/json`. JSON filenames use the extension `.json`.

Douglas Crockford originally specified the JSON format in the early 2000s. JSON was first standardized in 2013, as ECMA-404.[7] RFC 8259 (https://tools.ietf.org/html/rfc8259), published in 2017, is the current version of the Internet Standard STD 90 (https://tools.ietf.org/html/std90), and it remains consistent with ECMA-404.[8] That same year, JSON was also standardized as ISO/IEC 21778:2017.[1] The ECMA and ISO standards describe only the allowed syntax, whereas the RFC covers some security and interoperability considerations.[9]

Contents

History

JSON grew out of a need for stateless, real-time server-to-browser communication protocol without using browser plugins such as Flash or Java applets, the dominant methods used in the early 2000s.[10]

Douglas Crockford at the Yahoo Building. (2007)

Douglas Crockford first specified and popularized the JSON format.[11] The acronym originated at State Software, a company co-founded by Crockford and others in March 2001. The co-founders agreed to build a system that used standard browser capabilities and provided an abstraction layer for Web developers to create stateful Web applications that had a persistent duplex connection to a Web server by holding two Hypertext Transfer Protocol (HTTP) connections open and recycling them before standard browser time-outs if no further data were exchanged. The co-founders had a round-table discussion and voted whether to call the data format JSML or JSON, as well as under what license type to make it available. Crockford added a clause to the JSON license stating that "The Software shall be used for Good, not Evil," in order to open-source the JSON libraries while mocking corporate lawyers and those who are overly pedantic. Chip Morningstar developed the idea for the State Application Framework at State Software.[12][13] On the other hand, this clause led to license compatibility problems of the JSON license with other open-source licenses.[14]

A precursor to the JSON libraries was used in a children's digital asset trading game project named Cartoon Orbit at Communities.com (the State co-founders had all worked at this company previously) for Cartoon Network, which used a browser side plug-in with a proprietary messaging format to manipulate DHTML elements (this system is also owned by 3DO). Upon discovery of early Ajax capabilities, digiGroups, Noosh, and others used frames to pass information into the user browsers' visual field without refreshing a Web application's visual context, realizing real-time rich Web applications using only the standard HTTP, HTML and JavaScript capabilities of Netscape 4.0.5+ and IE 5+. Crockford then found that JavaScript could be used as an object-based messaging format for such a system. The system was sold to Sun Microsystems, Amazon.com and EDS. The JSON.org[15] website was launched in 2002. In December 2005, Yahoo! began offering some of its Web services in JSON.[16]

JSON was based on a subset of the JavaScript scripting language (specifically, Standard ECMA-262 3rd Edition—December 1999[17]) and is commonly used with JavaScript, but it is a language-independent data format. Code for parsing and generating JSON data is readily available in many programming languages. JSON's website lists JSON libraries by language.

Though JSON was originally advertised and believed to be a strict subset of JavaScript and ECMAScript,[18] it inadvertently allows some unescaped characters in strings that were illegal in JavaScript and ECMAScript string literals. JSON is a strict subset of ECMAScript as of the language's 2019 revision.[19][20] See Data portability issues below.

In October 2013, Ecma International published the first edition of its JSON standard ECMA-404.[7] That same year, RFC 7158 (https://tools.ietf.org/html/rfc7158) used ECMA-404 as a reference. In 2014, RFC 7159 (https://tools.ietf.org/html/rfc7159) became the main reference for JSON's Internet uses, superseding RFC 4627 (https://tools.ietf.org/html/rfc4627) and RFC 7158 (https://tools.ietf.org/html/rfc7158) (but preserving ECMA-262 and ECMA-404 as main references). In November 2017, ISO/IEC JTC 1/SC 22 published ISO/IEC 21778:2017[1] as an international standard. On 13 December 2017, the Internet Engineering Task Force obsoleted RFC 7159 (https://tools.ietf.org/html/rfc7159) when it published RFC 8259 (https://tools.ietf.org/html/rfc8259), which is the current version of the Internet Standard STD 90.[21][22]

Data types and syntax

JSON's basic data types are:

- Number: a signed decimal number that may contain a fractional part and may use exponential E notation, but cannot include non-numbers such as NaN. The format makes no distinction between integer and floating-point. JavaScript uses a double-precision floating-point format for all its numeric values, but other languages implementing JSON may encode numbers differently.
- String: a sequence of zero or more Unicode characters. Strings are delimited with double-quotation marks and support a backslash escaping syntax.
- Boolean: either of the values `true` or `false`
- Array: an ordered list of zero or more values, each of which may be of any type. Arrays use square bracket notation with comma-separated elements.
- Object: an unordered collection of name–value pairs where the names (also called keys) are strings. Objects are intended to represent associative arrays,[7] where each key is unique within an object. Objects are delimited with curly brackets and use commas to separate each pair, while within each pair the colon ':' character separates the key or name from its value.
- `null`: an empty value, using the word `null`

Whitespace is allowed and ignored around or between syntactic elements (values and punctuation, but not within a string value). Four specific characters are considered whitespace for this purpose: space, horizontal tab, line feed, and carriage return. In particular, the byte order mark must not be generated by a conforming implementation (though it may be accepted when parsing JSON). JSON does not provide syntax for comments.[23]

Early versions of JSON (such as specified by RFC 4627 (https://tools.ietf.org/html/rfc4627)) required that a valid JSON text must consist of only an object or an array type, which could contain other types within them. This restriction was dropped in RFC 7158 (https://tools.ietf.org/html/rfc7158), where a JSON text was redefined as any serialized value.

Example

The following example shows a possible JSON representation describing a person.

```json
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

Data portability issues

Although Douglas Crockford originally asserted that JSON is a strict subset of JavaScript, his specification actually allows valid JSON documents that are not valid JavaScript; JSON allows the Unicode line terminators `U+2028` LINE SEPARATOR and `U+2029` PARAGRAPH SEPARATOR to appear unescaped in quoted strings, while ECMAScript 2018 and older does not.[24][25] This is a consequence of JSON disallowing only "control characters". For maximum portability, these characters should be backslash-escaped. This subtlety is important when generating JSONP.

JSON exchange in an open ecosystem must be encoded in UTF-8.[8] The encoding supports the full Unicode character set, including those characters outside the Basic Multilingual Plane (U+10000 to U+10FFFF). However, if escaped, those characters must be written using UTF-16 surrogate pairs, a detail missed by some JSON parsers. For example, to include the Emoji character `U+1F610` 😐 NEUTRAL FACE in JSON:

```
{ "face": "😐" }
// or
{ "face": "\uD83D\uDE10" }
```

Numbers in JSON are agnostic with regard to their representation within programming languages. While this allows for numbers of arbitrary precision to be serialized, it may lead to portability issues. For example, since no differentiation is made between integer and floating-point values, some implementations may treat `42`, `42.0`, and `4.2E+1` as the same number, while others may not. The JSON standard makes no requirements regarding implementation details such as overflow, underflow, loss of precision, rounding, or signed zeros, but it does recommend to expect no more than IEEE 754 binary64 precision for "good interoperability". There is no inherent precision loss in serializing a machine-level binary representation of a floating-point number (like binary64) into a human-readable decimal representation (like numbers in JSON), and back, since there exist published algorithms to do this exactly and optimally.[26]

Comments were purposefully excluded from JSON. In 2012, Douglas Crockford described his design decision thus: "I removed comments from JSON because I saw people were using them to hold parsing directives, a practice which would have destroyed interoperability." [23]

Semantics

While JSON provides a syntactic framework for data interchange, unambiguous data interchange also requires agreement between producer and consumer on the semantics of a specific use of the JSON syntax.[27] One example of where such an agreement is necessary is the serialization of data types defined by the JavaScript syntax that are not part of the JSON standard, e.g. Date, Function, Regular Expression, and `undefined`.[note 2][28]

Metadata and schema

MIME type

The official MIME type for JSON text is "`application/json`",[29] and most modern implementations have adopted this.

The unofficial MIME type "`text/json`" or the content-type "`text/javascript`" also get legacy support by many service providers, browsers, servers, web applications, libraries, frameworks, and APIs. Notable examples include the Google Search API,[30] Yahoo!,[30][31] Flickr,[30] Facebook API,[32] Lift framework,[33] Dojo Toolkit 0.4,[34] etc.

JSON Schema

JSON Schema specifies a JSON-based format to define the structure of JSON data for validation, documentation, and interaction control. It provides a contract for the JSON data required by a given application, and how that data can be modified.[35]

JSON Schema is based on the concepts from XML Schema (XSD), but is JSON-based. As in XSD, the same serialization/deserialization tools can be used both for the schema and data; and is self-describing. It is specified in an Internet Draft at the IETF, currently in 2019-09 draft, which was released on September 19, 2019.[36] There are several validators available for different programming languages,[37] each with varying levels of conformance.

There is no standard filename extension, but some have suggested `.schema.json`.[38]

Example JSON Schema (2019-09):

```json
{
  "$schema": "http://json-schema.org/draft/2019-09/schema",
  "title": "Product",
  "type": "object",
  "required": ["id", "name", "price"],
  "properties": {
    "id": {
      "type": "number",
      "description": "Product identifier"
    },
    "name": {
      "type": "string",
      "description": "Name of the product"
    },
    "price": {
      "type": "number",
      "minimum": 0
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string"
      }
    },
    "stock": {
      "type": "object",
      "properties": {
        "warehouse": {
          "type": "number"
        },
        "retail": {
          "type": "number"
        }
      }
    }
  }
}
```

The JSON Schema above can be used to test the validity of the JSON text below:

```json
{
  "id": 1,
  "name": "Foo",
  "price": 123,
  "tags": [
    "Bar",
    "Eek"
  ],
  "stock": {
    "warehouse": 300,
    "retail": 20
  }
}
```

Object references

The JSON standard does not support object references, but an IETF draft standard for JSON-based object references exists.[39] The Dojo Toolkit supports object references using standard JSON; specifically, the `dojox.json.ref` module provides support for several forms of referencing including circular, multiple, inter-message, and lazy referencing. Internally both do so by assigning a `"$ref"` key for such references and resolving it at parse-time; the IETF draft only specifies the URL syntax, but Dojo allows more.[40][41][42]

Alternatively, non-standard solutions exist such as the use of Mozilla JavaScript Sharp Variables. However this functionality became obsolete with JavaScript 1.8.5 and was removed in Firefox version 12.[43]

Applications

JSON-RPC

JSON-RPC is a remote procedure call (RPC) protocol built on JSON, as a replacement for XML-RPC or SOAP. It is a simple protocol that defines only a handful of data types and commands. JSON-RPC lets a system send notifications (information to the server that does not require a response) and multiple calls to the server that can be answered out of order. Example of a JSON-RPC 2.0 request and response using positional parameters.

```
--> {"jsonrpc": "2.0", "method": "subtract", "params": [42, 23], "id": 1}
<-- {"jsonrpc": "2.0", "result": 19, "id": 1}
```

AJAJ

Asynchronous JavaScript and JSON (or AJAJ) refers to the same dynamic web page methodology as Ajax, but instead of XML, JSON is the data format. AJAJ is a web development technique that provides for the ability of a webpage to request new data after it has loaded into the web browser. Typically it renders new data from the server in response to user actions on that webpage. For example, what the user types into a search box, client-side code then sends to the server, which immediately responds with a drop-down list of matching database items.

The following JavaScript code is an example of a client using XMLHttpRequest to request data in JSON format from a server. (The server-side programming is omitted; it must be set up to service requests to the `url` containing a JSON-formatted string.)

```
var my_JSON_object;
var http_request = new XMLHttpRequest();
http_request.open("GET", url, true);
http_request.responseType = "json";
http_request.onreadystatechange = function () {
  var done = 4, ok = 200;
  if (http_request.readyState === done && http_request.status === ok) {
    my_JSON_object = http_request.response;
  }
};
http_request.send(null);
```

As a configuration language

While JSON is a data serialization format, it has seen ad hoc usage as a configuration language. In this use case, support for comments and other features have been deemed useful, which has led to several nonstandard JSON supersets being created. Among them are HJSON,[44] HOCON, and JSON5 (which despite its name, isn't the fifth version of JSON).[45][46] The primary objective of version 1.2 of YAML was to make the nonstandard format a strict JSON superset.[47]

In 2012, Douglas Crockford had this to say about comments in JSON when used as a configuration language: "I know that the lack of comments makes some people sad, but it shouldn't. Suppose you are using JSON to keep configuration files, which you would like to annotate. Go ahead and insert all the comments you like. Then pipe it through JSMin before handing it to your JSON parser."[23]

Security considerations

JSON is intended as a data serialization format. However, its design as a subset of JavaScript can lead to the misconception that it is safe to pass JSON texts to the JavaScript `eval()` function. This is not safe, due to the fact that certain valid JSON texts, specifically those containing `U+2028` or `U+2029`, are actually not valid JavaScript code.[48]

To avoid the many pitfalls caused by executing arbitrary code from the Internet, a new function, `JSON.parse()` was first added to the fifth edition of ECMAScript,[49] which as of 2017 is supported by all major browsers. For non-supported browsers, an API-compatible JavaScript library is provided by Douglas Crockford.[50] In addition, the TC39 proposal "Subsume JSON" (https://github.com/tc39/proposal-json-superset) made ECMAScript a strict JSON superset as of the language's 2019 revision.[19][20]

Various JSON parser implementations have suffered from denial-of-service attack and mass assignment vulnerability.[51][52]

Comparison with other formats

JSON is promoted as a low-overhead alternative to XML as both of these formats have widespread support for creation, reading, and decoding in the real-world situations where they are commonly used.[53] Apart from XML, examples could include CSV and YAML (a superset of JSON). Also, Google Protocol Buffers can fill this role, although it is not a data interchange language.

YAML

YAML version 1.2 is a superset of JSON; prior versions were not strictly compatible. For example, escaping a slash (/) with a backslash (\) is valid in JSON, but was not valid in YAML.[47] Such escaping is common practice when injecting JSON into HTML to protect against cross-site scripting attacks.

YAML supports comments, while JSON does not.[47][45][23]

XML

XML has been used to describe structured data and to serialize objects. Various XML-based protocols exist to represent the same kind of data structures as JSON for the same kind of data interchange purposes. Data can be encoded in XML in several ways. The most expansive form using tag pairs results in a much larger representation than JSON, but if data is stored in attributes and 'short tag' form where the closing tag is replaced with '/>', the representation is often about the same size as JSON or just a little larger. However, an XML attribute can only have a single value and each attribute can appear at most once on each element.

XML separates "data" from "metadata" (via the use of elements and attributes), while JSON does not have such a concept.

Another key difference is the addressing of values. JSON has objects with a simple "key" → "value" mapping, whereas in XML addressing happens on "nodes", which all receive a unique ID via the XML processor. Additionally, the XML standard defines a common attribute "xml:id", that can be used by the user, to set an ID explicitly.

XML tag names cannot contain any of the characters !"#$%&'()*+,/;<=>?@[\]^`{|}~, nor a space character, and cannot begin with "-", ".", or a numeric digit, whereas JSON keys can (even if quotation mark and reverse solidus must be escaped).[54]

XML values are strings of *characters*, with no built-in type safety. XML has the concept of schema, that permits strong typing, user-defined types, predefined tags, and formal structure, allowing for formal validation of an XML stream. JSON has strong typing built-in, and has a similar schema concept in JSON Schema.

XML supports comments, while JSON does not.[55][23]

Samples

JSON sample

```
{
  "first name": "John",
  "last name": "Smith",
  "age": 25,
  "address": {
    "street address": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postal code": "10021"
  },
  "phone numbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "fax",
      "number": "646 555-4567"
    }
  ],
  "sex": {
    "type": "male"
  }
}
```

Both of the following examples carry the same kind of information as the JSON example above in different ways, except that XML doesn't carry the data type.

YAML sample

The JSON text above is also entirely valid YAML. YAML also offers an alternative syntax intended to be more human-accessible by replacing nested delimiters like {}, [], and " marks with indentation.[47]

```
first name: John
last name: Smith
age: 25
address:
  street address: 21 2nd Street
  city: New York
  state: NY
  postal code: '10021'
phone numbers:
  - type: home
    number: 212 555-1234
  - type: fax
    number: 646 555-4567
sex:
  type: male
```

XML samples

Note that the XML examples below don't encode the data type (e.g. that age is a number), and would need something like a schema to encode the same information as the JSON example above does as is. Note that since XML tag names cannot contain spaces (in contrast to JSON and YAML keys), another naming convention has to be used to represent the JSON keys (in this example the tags are in camelCase).[54]

```xml
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>21 2nd Street</streetAddress>
    <city>New York</city>
    <state>NY</state>
    <postalCode>10021</postalCode>
  </address>
  <phoneNumbers>
    <phoneNumber>
      <type>home</type>
      <number>212 555-1234</number>
    </phoneNumber>
    <phoneNumber>
      <type>fax</type>
      <number>646 555-4567</number>
    </phoneNumber>
  </phoneNumbers>
  <sex>
    <type>male</type>
  </sex>
</person>
```

The properties can also be serialized using attributes instead of tags:

```xml
<person firstName="John" lastName="Smith" age="25">
  <address streetAddress="21 2nd Street" city="New York" state="NY" postalCode="10021" />
  <phoneNumbers>
    <phoneNumber type="home" number="212 555-1234"/>
    <phoneNumber type="fax" number="646 555-4567"/>
  </phoneNumbers>
  <sex type="male"/>
</person>
```

See also

- JSON streaming – on delimiting JSON in stream protocols
- Comparison of data serialization formats
- Related formats
  - GeoJSON – an open format for encoding a variety of geographic data structures
  - JSON-LD – JavaScript object notation for linked data, a W3C recommendation
  - JSON-RPC – remote procedure call protocol encoded in JSON
  - JsonML – markup language used to map between XML and JSON
  - S-expression – the comparable LISP format for trees as text.
  - SOAPjr – a hybrid of SOAP and JR (JSON-RPC)
  - GBSON - annotation format of nucleic acid sequences (DNA and RNA)[56]
- Related binary encodings
  - BSON – with data types of specific interest to MongoDB
  - CBOR – loosely based on JSON
  - MessagePack – with data types loosely corresponding with JSON's

- Smile – based on JSON
- UBJSON – binary format directly imitating JSON
- EXI4JSON (EXI for JSON) – representation by means of the Efficient XML Interchange (EXI) standard

- Implementations:
  - Jackson – JSON processor for Java

Notes

1. The 2017 international standard (ECMA-404 and ISO/IEC 21778:2017) specifies 'Pronounced /ˈdʒeɪ·sən/, as in "Jason and The Argonauts"'.[1][2] The first (2013) edition of ECMA-404 did not address the pronunciation.[3] The *UNIX and Linux System Administration Handbook* says 'Douglas Crockford, who named and promoted the JSON format, says it's pronounced like the name Jason. But somehow, "JAY-sawn" seems to have become more common in the technical community.'[4] Crockford said in 2011, "There's a lot of argument about how you pronounce that, but I strictly don't care."[5]
2. The `undefined` type was left out of the JSON standard, and one finds suggestions that `null` be used instead. In fact, the current standard says that for a sparse array such as:

   ```
   var v = [0];
   v[3] = 3;
   ```

   which behaves in JavaScript as if it were:

   ```
   var vx = [0, undefined, undefined, 3];
   ```

   with the `undefined` entries being only implicit rather than explicit, should translate to JSON as if it were:

   ```
   var vx = [0, null, null, 3];
   ```

   with explicit `null` fillers for the undefined entries.

   Furthermore, in JavaScript `{a: undefined}` often behaves the same as `{}`. Both translate as `"{}"` in JSON. However `undefined` as an explicit property value does have use in JavaScript inheritance situations such as:

   ```
   var x = {a: 1};
   var xi = Object.create(x);
   xi.a = undefined;
   ```

   where the inheritance of x's property a is overridden in `xi` and makes it pretty much behave as if nothing was inherited. `JSON.stringify` itself ignores inherited values - it only translates the enumerable own properties as given by `Object.keys(y)`. The default stringification, while not encoding inheritance, can (except for `undefined` values) encode enough of an object to reconstruct it in an environment that knows what inheritance it should have. To encode JavaScript objects that contain explicit `undefined` values a convention for representing `undefined` must be established, such as mapping it to the string `"UNDEFINED"`. One can then pass `JSON.stringify` the optional `replacer` argument to translate with this convention:

   ```
   var y = {a: undefined};
   var ys = JSON.stringify(y, function (k, v) { return (v === undefined) ? "UNDEFINED" : v });
   ```

Converting this JSON back into JavaScript is not as straightforward. While `JSON.parse` can take an optional `reviver` argument that is, essentially, the inverse of a `replacer`, it can't be used in this situation. If that function returns `undefined`, the `JSON.parse` logic interprets this to mean to not define a property rather than define one with a `undefined` value. Instead one has to explicitly post process the result from `JSON.parse` replacing each `"UNDEFINED"` with `undefined`.

References

1. 14:00-17:00. "ISO/IEC 21778:2017" (http://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/07/16/71616.html). *ISO*. Retrieved 29 July 2019.
2. "Standard ECMA-404 - The JSON Data Interchange Syntax" (https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf) (PDF). Ecma International. December 2017. p. 1, footnote. Retrieved 27 October 2019.
3. *ECMA-404: The JSON Data Interchange Format* (https://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-404%201st%20edition%20October%202013.pdf) (PDF) (1st ed.). Geneva: ECMA International. October 2013.
4. Nemeth, Evi; Snyder, Garth; Hein, Trent R.; Whaley, Ben; Mackin, Dan (2017). "19: Web Hosting" (https://books.google.com/books?id=f7M1DwAAQBAJ&pg=PT1125). *UNIX and Linux System Administration Handbook* (5th ed.). Addison-Wesley Professional. ISBN 9780134278292. Retrieved 29 October 2019.
5. "Douglas Crockford: The JSON Saga - Transcript Vids" (http://transcriptvids.com/v/-C-JoyNuQJs.html). *transcriptvids.com*. Retrieved 29 October 2019.
6. "A Modern Reintroduction To AJAX" (http://www.javascript-coder.com/tutorials/re-introduction-to-ajax.phtml). Retrieved 12 April 2017.
7. "The JSON Data Interchange Format" (https://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-404%201st%20edition%20October%202013.pdf) (PDF). ECMA International. October 2013. Retrieved 24 October 2019.
8. "The JavaScript Object Notation (JSON) Data Interchange Format" (https://tools.ietf.org/html/rfc8259). IETF. December 2017. Retrieved 16 February 2018.
9. Bray, Tim. "JSON Redux AKA RFC8259" (https://www.tbray.org/ongoing/When/201x/2014/03/05/RFC7159-JSON). *Ongoing*. Retrieved 16 March 2014.
10. "Unofficial Java History" (https://web.archive.org/web/20140526235903/http://www.edu4java.com/en/java/unofficial-java-history.html). *Edu4Java*. 26 May 2014. Archived from the original (http://www.edu4java.com/en/java/unofficial-java-history.html) on 26 May 2014. Retrieved 30 August 2019. "In 1996, Macromedia launches Flash technology which occupies the space left by Java and ActiveX, becoming the de facto standard for animation on the client side."
11. "Douglas Crockford — The JSON Saga" (https://www.youtube.com/watch?v=-C-JoyNuQJs). YouTube. 28 August 2011. Retrieved 23 September 2016.
12. "Chip Morningstar Biography" (http://www.fudco.com/chip/resume.html). n.d.
13. "State Software Breaks Through Web App Development Barrier With State Application Framework: Software Lets Developers Create Truly Interactive Applications; Reduces Costs, Development Time and Improves User Experience" (https://web.archive.org/web/20130605095712/http://www.prnewswire.com/news-releases/state-software-breaks-through-web-app-development-barrier-with-state-application-framework-75971782.html). *PR Newswire*. February 12, 2002. Archived from the original (http://www.prnewswire.com/news-releases/state-software-breaks-through-web-app-development-barrier-with-state-application-framework-75971782.html) on June 5, 2013. Retrieved March 19, 2013.
14. Apache and the JSON license (https://lwn.net/Articles/707510/) on LWN.net by Jake Edge (November 30, 2016)
15. "JSON" (http://json.org/). *json.org*.
16. Yahoo!. "Using JSON with Yahoo! Web services" (https://web.archive.org/web/20071011085815/http://developer.yahoo.com/common/json.html). Archived from the original (http://developer.yahoo.com/common/json.html) on October 11, 2007. Retrieved July 3, 2009.

17. Crockford, Douglas (May 28, 2009). "Introducing JSON" (http://json.org). json.org. Retrieved July 3, 2009. "It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999."
18. Douglas Crockford (2016-07-10). "JSON in JavaScript" (https://web.archive.org/web/20160710230817/http://www.json.org/js.html). Archived from the original on 2016-07-10. Retrieved 2016-08-13. "JSON is a subset of the object literal notation of JavaScript."
19. "Subsume JSON: Proposal to make all JSON text valid ECMA-262" (https://tc39.es/proposal-json-superset/). Ecma TC39. 23 August 2019. Retrieved 27 August 2019.
20. "Advance to Stage 4 - tc39/proposal-json-superset" (https://github.com/tc39/proposal-json-superset/commit/0604b6083e18fe033a1520388b8c6146bcd79e23). *GitHub*. May 22, 2018.
21. "History for draft-ietf-jsonbis-rfc7159bis-04" (https://datatracker.ietf.org/doc/rfc8259/history/). *IETF Datatracker*. Internet Engineering Task Force. Retrieved 2019-10-24. "2017-12-13 [...] RFC published"
22. "RFC 8259 - The JavaScript Object Notation (JSON) Data Interchange Format" (https://datatracker.ietf.org/doc/rfc8259/). *IETF Datatracker*. Internet Engineering Task Force. Retrieved 2019-10-24. "Type: RFC - Internet Standard (December 2017; Errata); Obsoletes RFC 7159; Also known as STD 90"
23. Crockford, Douglas (2012-04-30). "Comments in JSON" (https://archive.today/20150704102718/https://plus.google.com/+DouglasCrockfordEsq/posts/RK8qyGVaGSr). Archived from the original (https://plus.google.com/+DouglasCrockfordEsq/posts/RK8qyGVaGSr) on 2015-07-04. Retrieved 2019-08-30. "I removed comments from JSON because I saw people were using them to hold parsing directives, a practice which would have destroyed interoperability. I know that the lack of comments makes some people sad, but it shouldn't. Suppose you are using JSON to keep configuration files, which you would like to annotate. Go ahead and insert all the comments you like. Then pipe it through JSMin before handing it to your JSON parser."
24. Holm, Magnus (15 May 2011). "JSON: The JavaScript subset that isn't" (http://timelessrepo.com/json-isnt-a-javascript-subset). The timeless repository. Retrieved 23 September 2016.
25. "TC39 Proposal: Subsume JSON" (https://tc39.github.io/proposal-json-superset/). ECMA TC39 committee. 22 May 2018.
26. Andrysco, Marc; Jhala, Ranjit; Lerner, Sorin. "Printing Floating-Point Numbers - An Always Correct Method" (https://cseweb.ucsd.edu/~mandrysc/pub/dtoa.pdf) (PDF). Retrieved 2019-07-27.
27. "The JSON Data Interchange Syntax" (https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf) (PDF). Ecma International. December 2017. Retrieved 27 October 2019. "The JSON syntax is not a specification of a complete data interchange. Meaningful data interchange requires agreement between a producer and consumer on the semantics attached to a particular use of the JSON syntax. What JSON does provide is the syntactic framework to which such semantics can be attached"
28. "ECMAScript 2019 Language Specification" (https://web.archive.org/web/20150412040502/http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf) (PDF). Ecma International. June 2019. Archived from the original (http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf) (PDF) on 12 April 2015. Retrieved 27 October 2019.
29. "Media Types" (http://www.iana.org/assignments/media-types/application/index.html). *iana.org*. Retrieved 13 September 2015.
30. "Handle application/json & text/json by benschwarz · Pull Request #2 · mislav/faraday-stack" (https://github.com/mislav/faraday-stack/pull/2). *GitHub*. Retrieved 13 September 2015.
31. "Yahoo!, JavaScript, and JSON" (http://blog.programmableweb.com/2005/12/16/yahoo-javascript-and-json/). *ProgrammableWeb*. 2005-12-16. Retrieved 13 September 2015.
32. "Make JSON requests allow text/javascript content by jakeboxer · Pull Request #148 · AFNetworking/AFNetworking" (https://github.com/AFNetworking/AFNetworking/pull/148). *GitHub*. Retrieved 13 September 2015.
33. "lift/Req.scala at master · lift/lift · GitHub" (https://github.com/lift/lift/blob/master/framework/lift-base/lift-webkit/src/main/scala/net/liftweb/http/Req.scala). *GitHub*. Retrieved 13 September 2015.
34. "BrowserIO.js in legacy/branches/0.4/src/io – Dojo Toolkit" (https://web.archive.org/web/20160110132551/https://bugs.dojotoolkit.org/browser/legacy/branches/0.4/src/io/BrowserIO.js). *dojotoolkit.org*. Archived from the original (https://bugs.dojotoolkit.org/browser/legacy/branches/0.4/src/io/BrowserIO.js) on 10 January 2016. Retrieved 13 September 2015.
35. "JSON Schema and Hyper-Schema" (https://json-schema.org//). *json-schema.org*. Retrieved 11 February 2020.

36. "draft-handrews-json-schema-02 - JSON Schema: A Media Type for Describing JSON Documents" (https://json-schema.org/latest/json-schema-core.html). *json-schema.org/*. 2019-09-19. Retrieved 11 February 2020.
37. "JSON Schema Implementations" (https://json-schema.org/implementations.html). *json-schema.org*. Retrieved 11 February 2020.
38. "Json Schema file extension" (https://stackoverflow.com/a/10507586). *Stack Overflow*.
39. Zyp, Kris (September 16, 2012). Bryan, Paul C. (ed.). "JSON Reference: draft-pbryan-zyp-json-ref-03" (http://tools.ietf.org/html/draft-pbryan-zyp-json-ref-03). *Internet Engineering Task Force*.
40. Zyp, Kris. "dojox.json.ref" (https://dojotoolkit.org/reference-guide/dojox/json/ref.html). *Dojo*.
41. Zyp, Kris (June 17, 2008). "JSON referencing in Dojo" (http://www.sitepen.com/blog/2008/06/17/json-referencing-in-dojo). *SitePen*. Retrieved July 3, 2009.
42. von Gaza, Tys (Dec 7, 2010). "JSON referencing in jQuery" (https://web.archive.org/web/20150507001842/http://nubuntu.org/json-referencing-jquery). *NUBUNTU*. Archived from the original (http://nubuntu.org/json-referencing-jquery) on May 7, 2015. Retrieved Dec 7, 2010.
43. "Sharp variables in JavaScript" (https://developer.mozilla.org/en/Sharp_variables_in_JavaScript). *Mozilla Developer Network*. April 4, 2015. Retrieved 21 April 2012.
44. Edelman, Jason; Lowe, Scott; Oswalt, Matt. *Network Programmability and Automation*. O'Reilly Media. "for data representation you can pick one of the following: YAML, YAMLEX, JSON, JSON5, HJSON, or even pure Python"
45. McCombs, Thayne. "Why JSON isn't a good configuration language" (https://www.lucidchart.com/techblog/2018/07/16/why-json-isnt-a-good-configuration-language/). Lucid Chart. Retrieved 15 June 2019.
46. "HOCON (Human-Optimized Config Object Notation)" (https://github.com/lightbend/config/blob/master/HOCON.md). *GitHub*. 2019-01-28. Retrieved 2019-08-28. "The primary goal is: keep the semantics (tree structure; set of types; encoding/escaping) from JSON, but make it more convenient as a human-editable config file format."
47. "YAML Ain't Markup Language (YAML™) Version 1.2" (http://www.yaml.org/spec/1.2/spec.html). *yaml.org*. Retrieved 13 September 2015.
48. "JSON: The JavaScript subset that isn't" (http://timelessrepo.com/json-isnt-a-javascript-subset). Magnus Holm. Retrieved 16 May 2011.
49. "ECMAScript Fifth Edition" (https://web.archive.org/web/20110414214458/http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf) (PDF). Archived from the original (http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf) (PDF) on April 14, 2011. Retrieved March 18, 2011.
50. "douglascrockford/JSON-js" (https://github.com/douglascrockford/JSON-js/blob/master/json2.js). *GitHub*. 2019-08-13.
51. "Denial of Service and Unsafe Object Creation Vulnerability in JSON (CVE-2013-0269)" (https://www.ruby-lang.org/en/news/2013/02/22/json-dos-cve-2013-0269/). Retrieved January 5, 2016.
52. "Microsoft .NET Framework JSON Content Processing Denial of Service Vulnerability" (http://tools.cisco.com/security/center/viewAlert.x?alertId=31048). Retrieved January 5, 2016.
53. "JSON: The Fat-Free Alternative to XML" (http://www.json.org/xml.html). json.org. Retrieved 14 March 2011.
54. "XML 1.1 Specification" (https://www.w3.org/TR/xml11/). World Wide Web Consortium. Retrieved 2019-08-26.
55. Saternos, Casimir (2014). *Client-server web apps with Javascript and Java*. p. 45. ISBN 9781449369316.
56. "GBSON - A new annotation file format based on JSON" (https://chlorobox.mpimp-golm.mpg.de/GBSON.html).

External links

- Official website (https://json.org/)
- "ECMA-404 JSON Data Interchange Format" (http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf) (PDF). ECMA Int'l.
- STD 90 (https://tools.ietf.org/html/std90), JSON Data Interchange Format