

JSON

From Wikipedia, the free encyclopedia
(Redirected from JavaScript Object Notation)

JSON (an acronym for **JavaScript Object Notation** pronounced /dʒɛɪsən/) is a lightweight text-based open standard designed for human-readable data interchange. It is derived from the JavaScript programming language for representing simple data structures and associative arrays, called objects. Despite its relationship to JavaScript, it is language-independent, with parsers available for most programming languages.

The JSON format was originally specified by Douglas Crockford, and is described in RFC 4627. The official Internet media type for JSON is `application/json`. The JSON filename extension is `.json`.

The JSON format is often used for serializing and transmitting structured data over a network connection. It is primarily used to transmit data between a server and web application, serving as an alternative to XML.

JSON

Filename extension	<code>.json</code>
Internet media type	<code>application/json</code>
Type of format	Data interchange
Extended from	JavaScript
Standard(s)	RFC 4627
Website	http://json.org/

Contents

- 1 History
- 2 Data types, syntax and example
- 3 Schema
- 4 MIME type
- 5 Use in Ajax
- 6 Lack of support for date issue
- 7 Security issues
 - 7.1 JavaScript eval()
 - 7.2 Native encoding and decoding in browsers
- 8 Comparison with other formats
 - 8.1 XML
- 9 Efficiency
- 10 Object references
- 11 See also
- 12 References
- 13 External links

History

Douglas Crockford was the first to specify and popularize the JSON format.^[1]

JSON was used at State Software, a company co-founded by Crockford, starting around 2001. The JSON.org website was launched in 2002. In December 2005, Yahoo! began offering some of its web services in JSON.^[2] Google started offering JSON feeds for its GData web protocol in December 2006.^[3]

Although JSON was based on a subset of the JavaScript programming language (specifically, Standard ECMA-262 3rd Edition—December 1999^[4]) and is commonly used with that language, it is considered to be a language-independent data format. Code for parsing and generating JSON data is readily available for a large variety of programming languages. json.org (<http://json.org/>) provides a comprehensive listing of existing JSON libraries, organized by language.

Data types, syntax and example

JSON's basic types are:

- Number (double precision floating-point format)
- String (double-quoted Unicode with backslash escaping)
- Boolean (`true` or `false`)
- Array (an ordered sequence of values, comma-separated and enclosed in square brackets)
- Object (a collection of key:value pairs, comma-separated and enclosed in curly braces; the key must be a string)
- `null`

The following example shows the JSON representation of an object that describes a person. The object has string fields for first name and last name, a number field for age, contains an object representing the person's address, and contains a list (an array) of phone number objects.

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address":
  {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021"
  },
  "phoneNumber":
  [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "fax",
      "number": "646 555-4567"
    }
  ]
}
```

Since JSON is a subset of JavaScript it is possible (but not recommended) to parse the JSON text into an object by invoking JavaScript's `eval()` function. For example, if the above JSON data is contained within a JavaScript string variable *contact*, one could use it to create the JavaScript object *p* like so:

```
var p = eval("(" + contact + ")");
```

The `contact` variable must be wrapped in parentheses to avoid an ambiguity in JavaScript's syntax.^[5]

The recommended way, however, is to use a JSON parser. Unless a client absolutely trusts the source of the text, or must parse and accept text which is not strictly JSON-compliant, one should avoid `eval()`. A correctly implemented JSON parser will accept only valid JSON, preventing potentially malicious code from running.

Modern browsers, such as Firefox 3.5 and Internet Explorer 8, include special features for parsing JSON. As native browser support is more efficient and secure than `eval()`, native JSON support is included in the recently-released Edition 5 of the ECMAScript standard.^[6]

Schema

There are several ways to verify the structure and data types inside a JSON object, much like an XML schema.

JSON Schema^[7] is a specification for a JSON-based format for defining the structure of JSON data. JSON Schema provides a contract for what JSON data is required for a given application and how it can be modified, much like what XML Schema provides for XML. JSON Schema is intended to provide validation, documentation, and interaction control of JSON data. JSON Schema is based on the concepts from XML Schema, RelaxNG, and Kwalify, but is intended to be JSON-based, so that JSON data in the form of a schema can be used to validate JSON data, the same serialization/deserialization tools can be used for the schema and data, and it can be self descriptive.

JSON Schema is still an IETF draft,^[8] but there are several validators currently available for different programming languages,^[9] each with varying levels of conformance. Currently the most complete and compliant JSON Schema validator available is JSV.^[10]

Example JSON Schema:

```

{
    "name": "Product",
    "properties": {
        "id": {
            "type": "number",
            "description": "Product identifier",
            "required": true
        },
        "name": {
            "description": "Name of the product",
            "type": "string",
            "required": true
        },
        "price": {
            "type": "number",
            "minimum": 0,
            "required": true
        },
        "tags": {
            "type": "array",
            "items": {
                "type": "string"
            }
        }
    }
}

```

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

```

{
    "id": 1,
    "name": "Foo",
    "price": 123,
    "tags": [ "Bar", "Eek" ]
}

```

MIME type

The MIME type for JSON text is "application/json".

Use in Ajax

The following JavaScript code shows how the client can use an XMLHttpRequest to request an object in JSON format from the server. (The server-side programming is omitted; it has to be set up to respond to requests at url with a JSON-formatted string.)

```
var my_JSON_object = {};  
var http_request = new XMLHttpRequest();  
http_request.open( "GET", url, true );  
http_request.onreadystatechange = function () {  
    if (http_request.readyState == 4 && http_request.status == 200  
        my_JSON_object = JSON.parse( http_request.responseText );  
    }  
};  
http_request.send(null);
```

Note that the use of XMLHttpRequest in this example is not cross-browser compatible; syntactic variations are available for Internet Explorer, Opera, Safari, and Mozilla-based browsers. The usefulness of XMLHttpRequest is limited by the same origin policy: the URL replying to the request must reside within the same DNS domain as the server that hosts the page containing the request. Alternatively, the JSONP approach incorporates the use of an encoded callback function passed between the client and server to allow the client to load JSON-encoded data from third-party domains and to notify the caller function upon completion, although this imposes some security risks and additional requirements upon the server.

Browsers can also use <iframe> elements to asynchronously request JSON data in a cross-browser fashion, or use simple <form action="url_to_cgi_script" target="name_of_hidden_iframe"> submissions. These approaches were prevalent prior to the advent of widespread support for XMLHttpRequest.

Dynamic <script> tags can also be used to transport JSON data. With this technique it is possible to get around the same origin policy but it is insecure. JSONRequest (<http://json.org/JSONRequest.html>) has been proposed as a safer alternative.

Lack of support for date issue

Because date object cannot be expressed by a string literal according to JavaScript standard, date values cannot be converted to JSON string according to JSON standard (cf lack of mention of date in RFC4627). There are many attempts to workaround this issue, but as of 2011, there is still no consensus.

Security issues

Although JSON is intended as a data serialization format, its design as a subset of the JavaScript programming language poses several security concerns. These concerns center on the use of a JavaScript interpreter to execute JSON text dynamically as JavaScript, thus exposing a program to errant or malicious script contained therein—often a chief concern when dealing with data retrieved from the Internet. While not the only way to process JSON, it is an easy and popular technique, stemming from JSON's compatibility with JavaScript's eval() function, and illustrated by the following code examples.

JavaScript eval()

Because all JSON-formatted text is also syntactically legal JavaScript code, an easy way for a JavaScript program to parse JSON-formatted data is to use the built-in JavaScript `eval()` function, which was designed to evaluate JavaScript expressions. Rather than using a JSON-specific parser, the JavaScript interpreter itself is used to *execute* the JSON data to produce native JavaScript objects.

Unless precautions are taken to validate the data first, the `eval` technique is subject to security vulnerabilities if the data and the entire JavaScript environment is not within the control of a single trusted source. If the data is itself not trusted, for example, it may be subject to malicious JavaScript code injection attacks. Also, such breaches of trust may create vulnerabilities for data theft, authentication forgery, and other potential misuse of data and resources. Regular expressions can be used to validate the data prior to invoking `eval()`. For example, the RFC that defines JSON (RFC 4627) suggests using the following code to validate JSON before eval'ing it (the variable 'text' is the input JSON):^[11]

```
var my_JSON_object = !(/^[^,:{}\\\0-9.\-+Eaeflnr-u \n\r\t]/.test
    text.replace(/"(\.|\["\\])*"/g, '')) &&
    eval('(' + text + ')');
```

A new function, `JSON.parse()`, was developed as a safer alternative to `eval`. It is specifically intended to process JSON data and not JavaScript. It was originally planned for inclusion in the Fourth Edition of the ECMAScript standard,^[12] but this did not occur. It was first added to the Fifth Edition,^[13] and is now supported by the major browsers given below. For older ones, a compatible JavaScript library is available at JSON.org.

Native encoding and decoding in browsers

Recent Web browsers now either have or are working on native JSON encoding/decoding. This removes the `eval()` security problem above and also makes it faster because it doesn't parse functions. Native JSON is generally faster compared to the JavaScript libraries commonly used before. As of June 2009 the following browsers have or will have native JSON support:

- Mozilla Firefox 3.5+^[14]
- Microsoft Internet Explorer 8^[15]
- Opera 10.5+^[16]
- Webkit-based browsers (e.g. Google Chrome, Apple Safari)^[17]

At least 5 popular JavaScript libraries have committed to use native JSON if available:

- Yahoo! UI Library^[18]
- Prototype^[19]
- jQuery^[20]
- Dojo Toolkit^[21]
- mootools^[22]

Comparison with other formats

There are other lightweight markup languages that could be used to carry or store the same information payloads as JSON commonly does. Apart from XML, examples could include OGD, YAML, CSV and HTML. 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.^[23]

XML

XML is often 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. When data is encoded in XML, the result is typically larger in size than an equivalent encoding in JSON, mainly because of XML's closing tags. However, there are alternative ways to encode the same information. Each of the following XML examples carry the same information as the JSON example above in different ways.

```
<Object>
  <Property><Key>firstName</Key>      <String>John</String></Prop
  <Property><Key>lastName</Key>       <String>Smith</String></Pro
  <Property><Key>age</Key>             <Number>25</Number></Proper
  <Property><Key>address</Key>
    <Object>
      <Property><Key>streetAddress</Key> <String>21 2nd Street</
      <Property><Key>city</Key>          <String>New York</Strin
      <Property><Key>state</Key>         <String>NY</String></Pr
      <Property><Key>postalCode</Key>    <String>10021</String><
    </Object>
  </Property>
  <Property><Key>phoneNumber</Key>
    <Array>
      <Object>
        <Property><Key>type</Key>      <String>home</String>
        <Property><Key>number</Key>    <String>212 555-1234<
      </Object>
      <Object>
        <Property><Key>type</Key>      <String>fax</String><
        <Property><Key>number</Key>    <String>646 555-4567<
      </Object>
    </Array>
  </Property>
</Object>
```



```


<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>
  <phoneNumber type="home">212 555-1234</phoneNumber>
  <phoneNumber type="fax">646 555-4567</phoneNumber>
</person>

```

```

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

```



The XML encoding *may* therefore be shorter than the equivalent JSON encoding. A wide range of XML processing technologies exist, from the Document Object Model to XPath and XSLT. XML can also be styled for immediate display using CSS. XHTML is a form of XML so that elements can be passed in this form ready for direct insertion into webpages using client-side scripting.

Efficiency

JSON is primarily used for communicating data over the Internet, but has certain inherent characteristics that may limit its efficiency for this purpose. Most of the limitations are general limitations of textual data formats and also apply to XML and YAML. For example, despite typically being generated by an algorithm (by machine), parsing must be accomplished on a character-by-character basis. Additionally, the standard has no provision for data compression, interning of strings, or object references. Compression can, of course, be applied to the JSON formatted data (but the decompressed output typically still requires further full parsing for recognizable keywords, tags and delimiters).

Object references

The JSON standard does not support object references, but the Dojo Toolkit illustrates how conventions can be adopted to support such references using standard JSON. Specifically, the `dojox.json.ref` (<http://dojotoolkit.org/api/1.5/dojox/json/ref>) module provides support for several forms of referencing including circular, multiple, inter-message, and lazy referencing.^{[24][25]}

See also

- JSONP - JSON with Padding, a pattern of usage commonly employed when retrieving JSON from a webpage
- BSON - binary JSON

- JSON Summary JSON
- GeoJSON
- JSON-RPC
- SOAPjr – a hybrid of SOAP and JR (JSON-RPC)
- JsonML
- Comparison of data serialization formats

References

1. ^ Video: Douglas Crockford — The JSON Saga (<http://developer.yahoo.com/yui/theater/video.php?v=crockford-json>) , on Yahoo! Developer Network. In the video Crockford states: "I do not claim to have invented JSON ... What I did was I found it, I named it, I described how it was useful. [and a few sentences later ...] So the idea's been around there for awhile. What I did was I gave it a specification, and a little website."
2. ^ Yahoo!. "Using JSON with Yahoo! Web services" (<http://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. <http://web.archive.org/web/20071011085815/http://developer.yahoo.com/common/json.html>. Retrieved July 3, 2009.
3. ^ Google. "Using JSON with Google Data APIs" (<http://code.google.com/apis/gdata/json.html>) . <http://code.google.com/apis/gdata/json.html>. Retrieved July 3, 2009.
4. ^ Crockford, Douglas (May 28, 2009). "Introducing JSON" (<http://json.org>) . json.org. <http://json.org>. Retrieved July 3, 2009.
5. ^ Crockford, Douglas (July 9, 2008). "JSON in JavaScript" (<http://www.json.org/js.html>) . [json.org](http://www.json.org/js.html). <http://www.json.org/js.html>. Retrieved September 8, 2008.
6. ^ <http://www.ecma-international.org/publications/standards/Ecma-262.htm>
7. ^ JSON Schema (<http://json-schema.org>)
8. ^ JSON Schema draft 3 (<http://tools.ietf.org/html/draft-zyp-json-schema-03>)
9. ^ JSON Schema implementations (<http://groups.google.com/group/json-schema/web/json-schema-implementations>)
10. ^ JSV: JSON Schema Validator (<http://github.com/garycourt/JSV>)
11. ^ Douglas Crockford (July 2006). "IANA Considerations" (<http://tools.ietf.org/html/rfc4627#section-6>) . *The application/json Media Type for JavaScript Object Notation (JSON)* (<http://tools.ietf.org/html/rfc4627>) . IETF. sec. 6. RFC 4627. <http://tools.ietf.org/html/rfc4627#section-6>. Retrieved October 21, 2009.
12. ^ Crockford, Douglas (December 6, 2006). "JSON: The Fat-Free Alternative to XML" (<http://www.json.org/fatfree.html>) . <http://www.json.org/fatfree.html>. Retrieved July 3, 2009.
13. ^ "ECMAScript Fifth Edition" (<http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf>) . <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf>. Retrieved March 18, 2011.
14. ^ "Using Native JSON" (https://developer.mozilla.org/en/Using_JSON_in_Firefox) . June 30, 2009. https://developer.mozilla.org/en/Using_JSON_in_Firefox. Retrieved July 3, 2009.
15. ^ Barsan, Corneliu (September 10, 2008). "Native JSON in IE8" (<http://blogs.msdn.com/ie/archive/2008/09/10/native-json-in-ie8.aspx>) . <http://blogs.msdn.com/ie/archive/2008/09/10/native-json-in-ie8.aspx>. Retrieved July 3, 2009.
16. ^ "Web specifications supported in Opera Presto 2.5" (<http://www.opera.com/docs/specs/presto25/#ecmascript>) . March 10, 2010. <http://www.opera.com/docs/specs/presto25/#ecmascript>. Retrieved March 29, 2010.
17. ^ Hunt, Oliver (June 22, 2009). "Implement ES 3.1 JSON object" (https://bugs.webkit.org/show_bug.cgi?id=20031) . https://bugs.webkit.org/show_bug.cgi?id=20031. Retrieved July 3, 2009.
18. ^ "YUI 2: JSON utility" (<http://developer.yahoo.com/yui/json/#native>) . September 1, 2009. <http://developer.yahoo.com/yui/json/#native>. Retrieved October 22, 2009.
19. ^ "Learn JSON" (<http://www.prototypejs.org/learn/json>) . April 7, 2010. <http://www.prototypejs.org/learn/json>. Retrieved April 7, 2010.
20. ^ "Ticket #4429" (<http://dev.jquery.com/ticket/4429>) . May 22, 2009. <http://dev.jquery.com/ticket/4429>. Retrieved July 3, 2009.
21. ^ "Ticket #8111" (<http://trac.dojotoolkit.org/ticket/8111>) . June 15, 2009. <http://trac.dojotoolkit.org/ticket/8111>. Retrieved July 3, 2009.
22. ^ "Ticket 419" (<https://mootools.lighthouseapp.com/projects/2706/tickets/419-use-the-native-json-object-if->

available) . October 11, 2008. <https://mootools.lighthouseapp.com/projects/2706/tickets/419-use-the-native-json-object-if-available>. Retrieved July 3, 2009.

23. ^ "JSON: The Fat-Free Alternative to XML" (<http://www.json.org/xml.html>) . json.org. <http://www.json.org/xml.html>. Retrieved 14 March 2011.
24. ^ Zyp, Kris (June 17, 2008). "JSON referencing in Dojo" (<http://www.sitepen.com/blog/2008/06/17/json-referencing-in-dojo>) . <http://www.sitepen.com/blog/2008/06/17/json-referencing-in-dojo>. Retrieved July 3, 2009.
25. ^ von Gaza, Tys (Dec 7, 2010). "JSON referencing in jQuery" (<http://plugins.jquery.com/project/jsonref>) . <http://plugins.jquery.com/project/jsonref>. Retrieved Dec 7, 2010.

External links

- Format home page (<http://www.json.org/>)
- RFC 4627, current formal JSON specification (<http://www.ietf.org/rfc/rfc4627.txt>)
- JSON-Introduction By Microsoft (<http://msdn.microsoft.com/en-us/library/bb299886.aspx>)
- JSON Schema Proposal (<http://groups.google.com/group/json-schema/web/json-schema-proposal-working-draft>)
- Limitations of JSON (http://blogs.sun.com/bb1fish/entry/the_limitations_of_json)
- JSON Questions and Answer (<http://www.pcds.co.in/json-interview-questions-and-answer.php>)
- Mastering JSON ([http://www.hunlock.com/blogs/Mastering_JSON_\(JavaScript_Object_Notation_\)](http://www.hunlock.com/blogs/Mastering_JSON_(JavaScript_Object_Notation_)))
- Relationship between JSON and YAML (<http://redhanded.hobix.com/inspect/yamlIsJson.html>)
- wxJSON - The wxWidgets implementation of JSON (<http://wxcode.sourceforge.net/docs/wxjson/>)
- XSLT and XPath for JSON (<http://www.p6r.com/articles/2008/05/06/xslt-and-xpath-for-json/>)

Retrieved from "<http://en.wikipedia.org/wiki/JSON>"

Categories: Ajax (programming) | Data serialization formats | JavaScript | Markup languages

- This page was last modified on 14 April 2011 at 17:02.
 - Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. See Terms of Use for details.
- Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.