[Template:Redirect](/wiki/Template:Redirect" \o "Template:Redirect)

[Template:Infobox file format](/wiki/Template:Infobox_file_format)

**JSON** (canonically pronounced [Template:IPAc-en](/wiki/Template:IPAc-en) [Template:Respell](/wiki/Template:Respell);[[1]](#cite_note-1) sometimes **JavaScript Object Notation**) is an [open-standard](/wiki/Open_standard) [format](/wiki/File_format#Chunk-based_formats) that uses [human-readable](/wiki/Human-readable) text to transmit data objects consisting of [attribute–value pairs](/wiki/Attribute–value_pair). It is the most common data format used for asynchronous browser/server communication ([AJAJ](/wiki/AJAJ)), largely replacing [XML](/wiki/XML) which is used by [AJAX](/wiki/Ajax_(programming)).

JSON is a [language-independent](/wiki/Language-independent_specification) data format. It derives from [JavaScript](/wiki/JavaScript), but [Template:As of](/wiki/Template:As_of), code to generate and [parse](/wiki/Parse) JSON-format data is available in many [programming languages](/wiki/Programming_languages). The official [Internet media type](/wiki/Internet_media_type) for JSON is application/json. The JSON filename extension is .json.

[Douglas Crockford](/wiki/Douglas_Crockford) originally specified the JSON format; two competing standards, RFC 7159 and ECMA-404, define it. The ECMA standard describes only the allowed syntax, whereas the RFC also provides some semantic and security considerations.[[2]](#cite_note-2) A typical [mashup](/wiki/Mashup_(web_application_hybrid)) fetches JSON-format data from several different web servers using an [Open API](/wiki/Open_API).

## Contents

* 1 History[[edit](/index.php?title=(none)&action=edit&section=1)]
* 2 Schema and metadata[[edit](/index.php?title=(none)&action=edit&section=7)]
  + 2.1 JSON Schema[[edit](/index.php?title=(none)&action=edit&section=8)]
* 3 MIME type[[edit](/index.php?title=(none)&action=edit&section=9)]
* 4 Applications[[edit](/index.php?title=(none)&action=edit&section=10)]
  + 4.1 JSON-RPC[[edit](/index.php?title=(none)&action=edit&section=11)]
  + 4.2 AJAJ[[edit](/index.php?title=(none)&action=edit&section=12)]
* 5 Security issues[[edit](/index.php?title=(none)&action=edit&section=13)]
  + 5.1 JavaScript {{code|lang=javascript|code=eval()}}[[edit](/index.php?title=(none)&action=edit&section=14)]
  + 5.2 Native encoding and decoding in browsers[[edit](/index.php?title=(none)&action=edit&section=15)]
  + 5.3 Implementation-specific issues[[edit](/index.php?title=(none)&action=edit&section=16)]
* 6 Object references[[edit](/index.php?title=(none)&action=edit&section=17)]
* 7 Comparison with other formats[[edit](/index.php?title=(none)&action=edit&section=18)]
  + 7.1 YAML[[edit](/index.php?title=(none)&action=edit&section=19)]
  + 7.2 XML[[edit](/index.php?title=(none)&action=edit&section=20)]
  + 7.3 Samples[[edit](/index.php?title=(none)&action=edit&section=21)]
    - 7.3.1 JSON sample[[edit](/index.php?title=(none)&action=edit&section=22)]
    - 7.3.2 YAML sample[[edit](/index.php?title=(none)&action=edit&section=23)]
    - 7.3.3 XML samples[[edit](/index.php?title=(none)&action=edit&section=24)]
* 8 See also[[edit](/index.php?title=(none)&action=edit&section=25)]
* 9 Notes[[edit](/index.php?title=(none)&action=edit&section=26)]
* 10 References[[edit](/index.php?title=(none)&action=edit&section=27)]
* 11 External links[[edit](/index.php?title=(none)&action=edit&section=28)]

## History[[edit](/index.php?title=(none)&action=edit&section=1)]

JSON grew out of a need for [stateful](/wiki/State_(computer_science)), real-time server-to-browser communication without using browser plugins such as [Flash](/wiki/Adobe_Flash) or Java applets, which were the dominant methods in the early 2000s.

[Douglas Crockford](/wiki/Douglas_Crockford) was the first to specify and popularize the JSON format.[[3]](#cite_note-3) The acronym was coined at State Software, a company co-founded by Crockford, [Chip Morningstar](/wiki/Chip_Morningstar) and Robert F. Napiltonia in April 2001 and funded by Tesla Ventures. The co-founders agreed to build a system that used standard browser capabilities and provided an [abstraction layer](/wiki/Abstraction_layer) for Web developers to create stateful Web applications that had a persistent duplex connection to a Web server by holding two [HTTP](/wiki/Hypertext_Transfer_Protocol) connections open and recycling them before standard browser time-outs if no further data were exchanged. The idea for the State Application Framework was developed by Morningstar at State Software.[[4]](#cite_note-4)[[5]](#cite_note-5)JSON was used in a project at Communities.com for Cartoon Network, which used a plug-in with a proprietary messaging format to manipulate [DHTML](/wiki/Dynamic_HTML) elements (this system is also owned by 3DO). Upon discovery of early [Ajax](/wiki/Ajax_(programming)) 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](/wiki/Sun_Microsystems), [Amazon.com](/wiki/Amazon.com) and [EDS](/wiki/Electronic_Data_Systems). The [JSON.org](http://json.org/) Web site was launched in 2002. In December 2005, [Yahoo!](/wiki/Yahoo!) began offering some of its [Web services](/wiki/Web_service) in JSON.[[6]](#cite_note-6) [Google](/wiki/Google) started offering JSON feeds for its [GData](/wiki/GData) web protocol in December 2006.[[7]](#cite_note-7) JSON was originally based on a non-strict subset of the [JavaScript](/wiki/JavaScript) scripting language (specifically, Standard [ECMA](/wiki/Ecma_International)-262 3rd Edition—December 1999[[8]](#cite_note-8)) and is commonly used with Javascript, but it is a [language-independent](/wiki/Language-independent_specification) data format. Code for [parsing](/wiki/Parsing) and generating JSON data is readily available in many [programming languages](/wiki/Programming_languages). JSON's Web site lists JSON [libraries](/wiki/Language_binding) by language.

Though JSON is commonly perceived as being a subset of JavaScript and ECMAScript, it allows some unescaped characters in strings that are illegal in JavaScript and ECMAScript strings.[[9]](#cite_note-9) JSON itself became an ECMA international standard in 2013 as the *ECMA-404 standard*.[[10]](#cite_note-10) These JavaScript data types must be represented by some other data format, with the programs on both ends agreeing on how to convert between the types. [Template:As of](/wiki/Template:As_of), there are some de facto standards, *e.g.*, converting from Date to String, but none universally recognized.[[18]](#cite_note-18)[[19]](#cite_note-19) Other languages may have a different set of native types that must be [serialized](/wiki/Serialization) carefully to deal with this type of conversion.

## Schema and metadata[[edit](/index.php?title=(none)&action=edit&section=7)]

### JSON Schema[[edit](/index.php?title=(none)&action=edit&section=8)]

[Template:Anchor](/wiki/Template:Anchor) JSON Schema[[20]](#cite_note-20) specifies a JSON-based format to define the structure of JSON data for validation, documentation, and interaction control. A JSON Schema provides a contract for the JSON data required by a given application, and how that data can be modified.

JSON Schema is based on the concepts from [XML Schema](/wiki/XML_Schema_(W3C)) (XSD), but is JSON-based. The JSON data schema can be used to validate JSON data. As in XSD, the same serialization/deserialization tools can be used both for the schema and data. The schema is self-describing.

JSON Schema was an [Internet Draft](/wiki/Internet_Draft), most recently version 4, which expired on August 4, 2013.[[21]](#cite_note-21) There are several validators available for different programming languages,[[22]](#cite_note-22) each with varying levels of conformance.

Example JSON Schema (draft 4): <source lang="JSON"> {

"$schema": "<http://json-schema.org/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"

}

}

}

}

} </source>

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

<source lang="JSON"> {

"id": 1,

"name": "Foo",

"price": 123,

"tags": [

"Bar",

"Eek"

],

"stock": {

"warehouse": 300,

"retail": 20

}

} </source>

## MIME type[[edit](/index.php?title=(none)&action=edit&section=9)]

The official [MIME type](/wiki/MIME_type) for JSON text is "application/json".[[23]](#cite_note-23) Although most modern implementations have adopted the official MIME type, many applications continue to provide legacy support for other MIME types. Many service providers, browsers, servers, web applications, libraries, frameworks, and APIs use, expect, or recognize the (unofficial) MIME type "text/json" or the content-type "text/javascript". Notable examples include the Google Search API,[[24]](#cite_note-24) Yahoo!,[[24]](#cite_note-24)[[25]](#cite_note-25) Flickr,[[24]](#cite_note-24) Facebook API,[[26]](#cite_note-26) [Lift framework](/wiki/Lift_(web_framework)),[[27]](#cite_note-27) Dojo Toolkit 0.4,[[28]](#cite_note-28) etc.

## Applications[[edit](/index.php?title=(none)&action=edit&section=10)]

### JSON-RPC[[edit](/index.php?title=(none)&action=edit&section=11)]

[JSON-RPC](/wiki/JSON-RPC) is an [RPC](/wiki/Remote_procedure_call) protocol built on JSON, as a replacement for [XML-RPC](/wiki/XML-RPC) or [SOAP](/wiki/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. <source lang="javascript"> --> {"jsonrpc": "2.0", "method": "subtract", "params": [42, 23], "id": 1} <-- {"jsonrpc": "2.0", "result": 19, "id": 1} </source>

### AJAJ[[edit](/index.php?title=(none)&action=edit&section=12)]

[Asynchronous JavaScript and JSON](/wiki/Asynchronous_JavaScript_and_JSON) (or AJAJ) refers to the same [dynamic web page](/wiki/Dynamic_web_page) methodology as [Ajax](/wiki/Ajax_(programming)), but instead of [XML](/wiki/XML), JSON is the data format. AJAJ is a web development technique that provides to the ability of a [webpage](/wiki/Webpage) to request new data after it has loaded into the [web browser](/wiki/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](/wiki/Text_box), [client-side code](/wiki/Client-side_scripting) then sends to the server, which immediately responds with a [drop-down list](/wiki/Drop-down_list) of matching [database](/wiki/Database) items.

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

<source lang=JavaScript> 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); </source>

## Security issues[[edit](/index.php?title=(none)&action=edit&section=13)]

JSON is intended as a [data serialization](/wiki/Data_serialization) format. However, its design as a non-strict subset of the JavaScript scripting language poses several security concerns. These concerns center on the use of a JavaScript interpreter to execute JSON text dynamically as embedded JavaScript. This exposes a program to errant or malicious scripts. This is a serious issue when dealing with data retrieved from the Internet. This easy and popular but risky technique exploits JSON's compatibility with the JavaScript [Template:Code](/wiki/Template:Code) function, which is described below.

### JavaScript {{code|lang=javascript|code=eval()}}[[edit](/index.php?title=(none)&action=edit&section=14)]

Because most JSON-formatted text is also syntactically legal JavaScript code, a seductively easy way for a JavaScript program to parse JSON-formatted data is to use the built-in JavaScript [Template:Code](/wiki/Template:Code) function, which was designed to evaluate JavaScript [expressions](/wiki/Expression_(computer_science)). Rather than using a JSON-specific parser, the JavaScript interpreter itself is used to *execute* the JSON data producing native JavaScript objects. This technique is terribly risky, however, if there is any chance that the JSON data might contain arbitrary JavaScript code, which would then be executed also.

Unless precautions are taken to validate the data first, the eval technique is subject to [security vulnerabilities](/wiki/Vulnerability_(computing)) when the data and the entire JavaScript environment are not within the control of a single [trusted source](/wiki/Trusted_system). For example, if the data is itself not trusted, it is subject to malicious JavaScript [code injection](/wiki/Code_injection) attacks. Such breaches of trust also can create vulnerabilities for [data theft](/wiki/Data_theft), [authentication forgery](/wiki/Digital_identity), and other potential misuse of data and resources.

As a workaround, a [regular expression](/wiki/Regular_expression) can be used to partially validate the data prior to invoking [Template:Code](/wiki/Template:Code). The RFC that defines JSON (RFC 4627) suggests using the following code to validate JSON before evaluating it (the variable 'text' is the input JSON):[[29]](#cite_note-29) <source lang=JavaScript> var my\_JSON\_object = !(/[^,:{}\[\]0-9.\-+Eaeflnr-u \n\r\t]/.test(

text.replace(/"(\\.|[^"\\])\*"/g, *))) && eval('(' + text + ')');*

</source>

However, this validation is now known to be insufficient.[[30]](#cite_note-30) A new function, JSON.parse(), was thus 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](/wiki/ECMAScript) standard,[[31]](#cite_note-31) but this did not occur. It was first added to the Fifth Edition,[[32]](#cite_note-32) and is now supported by the major browsers given below. For older ones, a compatible JavaScript library is available at JSON.org.

An additional issue when parsing JSON using the [Template:Code](/wiki/Template:Code) function is that there are some Unicode characters that are valid in JSON strings but invalid in JavaScript, so additional escaping may be needed in some cases.[[33]](#cite_note-33)

### Native encoding and decoding in browsers[[edit](/index.php?title=(none)&action=edit&section=15)]

Web browsers now have or plan to have native JSON encoding and decoding. This eliminates the [Template:Code](/wiki/Template:Code) security problem above, and can increase performance compared to the JavaScript libraries commonly used before. As of June 2009, the following browsers have native JSON support, via JSON.parse() and JSON.stringify():

* [Mozilla Firefox 3.5](/wiki/Mozilla_Firefox_3.5)+[[34]](#cite_note-34)\* Microsoft [Internet Explorer](/wiki/Internet_Explorer) 8+[[35]](#cite_note-35)\* [Opera 10.5](/wiki/Opera_(web_browser))+[[36]](#cite_note-36)\* [WebKit](/wiki/WebKit)-based browsers (Apple Safari)[[37]](#cite_note-37)\*[Blink](/wiki/Blink_(layout_engine))-based browsers (e.g. Google Chrome, Opera)

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

* [YUI Library](/wiki/YUI_Library)[[38]](#cite_note-38)\* [Prototype](/wiki/Prototype_JavaScript_Framework)[[39]](#cite_note-39)\* [jQuery](/wiki/JQuery)[[40]](#cite_note-40)\* [Dojo Toolkit](/wiki/Dojo_Toolkit)[[41]](#cite_note-41)\* [MooTools](/wiki/MooTools)[[42]](#cite_note-42)

### Implementation-specific issues[[edit](/index.php?title=(none)&action=edit&section=16)]

Various JSON parser implementations suffered in the past from [denial-of-service attack](/wiki/Denial-of-service_attack) and [mass assignment vulnerability](/wiki/Mass_assignment_vulnerability).[[43]](#cite_note-43)[[44]](#cite_note-44)

## Object references[[edit](/index.php?title=(none)&action=edit&section=17)]

The JSON standard does not support object [references](/wiki/Reference_(computer_science)), but an [IETF](/wiki/IETF) draft standard for JSON-based object references exists <http://tools.ietf.org/html/draft-pbryan-zyp-json-ref-03>. The [Dojo Toolkit](/wiki/Dojo_Toolkit) supports object references using standard JSON; specifically, the [dojox.json.ref](https://dojotoolkit.org/reference-guide/dojox/json/ref.html) module provides support for several forms of referencing including [circular](/wiki/Circular_reference), multiple, inter-message, and [lazy](/wiki/Lazy_evaluation) referencing.[[45]](#cite_note-45)[[46]](#cite_note-46)Alternatively, non-standard solutions exist such as the use of Mozilla JavaScript Sharp Variables, although this functionality has been removed in Firefox version 12.[[47]](#cite_note-47)

## Comparison with other formats[[edit](/index.php?title=(none)&action=edit&section=18)]

[Template:See also](/wiki/Template:See_also)

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.[[48]](#cite_note-48) Apart from XML, examples could include [OGDL](/wiki/OGDL), [YAML](/wiki/YAML) and [CSV](/wiki/Comma-separated_values). Also, [Google Protocol Buffers](/wiki/Protocol_Buffers) can fill this role, although it is not a data interchange language.

### YAML[[edit](/index.php?title=(none)&action=edit&section=19)]

[YAML](/wiki/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. (This is common practice when injecting JSON into HTML to protect against [cross-site scripting attacks](/wiki/Cross-site_scripting).) Nonetheless, many YAML parsers can natively parse the output from many JSON encoders.[[49]](#cite_note-49)

### XML[[edit](/index.php?title=(none)&action=edit&section=20)]

[XML](/wiki/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 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 'slash-greater than', the representation is often about the same size as JSON or just a little larger. If the data is compressed using an algorithm like [gzip](/wiki/Gzip), there is little difference because compression is good at saving space when a pattern is repeated.

XML also has the concept of [schema](/wiki/XML_schema). This permits strong typing, user-defined types, predefined tags, and formal structure, allowing for formal validation of an XML stream in a portable way. There is, however, an [IETF](/wiki/IETF) draft proposal for a schema system for JSON <http://json-schema.org/documentation.html>.

XML supports comments, but JSON does not.<ref name=Saternos45>[Template:Cite book](/wiki/Template:Cite_book)</ref>

### Samples[[edit](/index.php?title=(none)&action=edit&section=21)]

#### JSON sample[[edit](/index.php?title=(none)&action=edit&section=22)]

<source lang="json"> {

"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"

}

],

"gender": {

"type": "male"

}

} </source> Both of the following examples carry the same kind of information as the JSON example above in different ways. There is a powerful processing technology to deal with JSON by the name of *jq*.[[50]](#cite_note-50)[[51]](#cite_note-51)[[52]](#cite_note-52)

#### YAML sample[[edit](/index.php?title=(none)&action=edit&section=23)]

The JSON code above is also entirely valid [YAML](/wiki/YAML). YAML also offers an alternative syntax intended to be more human-accessible by replacing nested delimiters like {}, [], and " marks with [off-side](/wiki/Off-side_rule) indentation.[[49]](#cite_note-49)<source lang=YAML> --- 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

gender:

type: male

</source>

#### XML samples[[edit](/index.php?title=(none)&action=edit&section=24)]

<source lang=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>

<gender>

<type>male</type>

</gender>

</person> </source>

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

<source lang=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>

<gender type="male"/>

</person> </source>

The XML encoding *may* therefore be comparable in length to the equivalent JSON encoding. A wide range of XML processing technologies exist, from the [Document Object Model](/wiki/Document_Object_Model) to [XPath](/wiki/XPath) and [XSLT](/wiki/XSLT). XML can also be styled for immediate display using [CSS](/wiki/Cascading_Style_Sheets). [XHTML](/wiki/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.

## See also[[edit](/index.php?title=(none)&action=edit&section=25)]

* [JSON Streaming](/wiki/JSON_Streaming)
* Other formats
  + [HOCON](/wiki/HOCON)—Human-Optimized Config Object Notation, a superset of JSON
  + [YAML](/wiki/YAML)—Another datastorage format that is a superset of JSON[[53]](#cite_note-53)\*\* [S-expression](/wiki/S-expression)—the comparable LISP format for trees as text.
  + [JSONP](/wiki/JSONP)—JSON with Padding, a pattern of usage commonly employed when retrieving JSON across domains
  + [GeoJSON](/wiki/GeoJSON)—an open format for encoding a variety of geographic data structures
  + [JSON-LD](/wiki/JSON-LD)—JavaScript Object Notation for Linked Data, currently a W3C Recommendation
  + [JSON-RPC](/wiki/JSON-RPC)
  + [SOAPjr](/wiki/SOAPjr)—a hybrid of SOAP and JR (JSON-RPC)
  + [JsonML](/wiki/JsonML)
* Binary encodings for JSON
  + [BSON](/wiki/BSON)
  + [MessagePack](/wiki/MessagePack)
  + [Smile](/wiki/Smile_(Data_Interchange_Format))
  + [UBJSON](/wiki/UBJSON)
* Implementations:
  + [Jayrock](/wiki/Jayrock)—an open source implementation of JSON for the [.NET Framework](/wiki/.NET_Framework).
  + [Ember data](/wiki/Ember.js) server implementations for PHP, Node.js, Ruby, Python, Go, .NET and Java.
* Other
  + [Comparison of data serialization formats](/wiki/Comparison_of_data_serialization_formats)

## Notes[[edit](/index.php?title=(none)&action=edit&section=26)]

[Template:Reflist](/wiki/Template:Reflist)

## References[[edit](/index.php?title=(none)&action=edit&section=27)]

[Template:Reflist](/wiki/Template:Reflist)

## External links[[edit](/index.php?title=(none)&action=edit&section=28)]

[Template:Sister project links](/wiki/Template:Sister_project_links)

* [Format home page](http://www.json.org/)
* [The JavaScript Object Notation (JSON) Data Interchange Format (RFC 7159)](https://tools.ietf.org/html/rfc7159)
* [ECMA-404](http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf)—The JSON Data Interchange Format
* [CBOR](http://tools.ietf.org/html/rfc7049)—A binary encoding for JSON
* [EXI for JSON](http://www.w3.org/TR/exi-for-json)—An efficient JSON representation by means of [EXI](http://www.w3.org/TR/exi/)
* ["Learn X in Y minutes" about JSON](https://learnxinyminutes.com/docs/json/)
* ["Yet another JSON Library" by](https://www.scottklement.com/yajl/) [Scott Klement](/wiki/Scott_Klement) brings JSON to [RPG](/wiki/IBM_RPG) and the [IBM i](/wiki/IBM_i)

[Template:Data Exchange](/wiki/Template:Data_Exchange)

[Template:Authority control](/wiki/Template:Authority_control)

[Category:2001 introductions](/wiki/Category:2001_introductions) [Category:Ajax (programming)](/wiki/Category:Ajax_(programming)) [Category:Data serialization formats](/wiki/Category:Data_serialization_formats) [Category:JavaScript](/wiki/Category:JavaScript) [Category:JSON](/wiki/Category:JSON) [Category:Markup languages](/wiki/Category:Markup_languages)