JSON

JSON (ˈdʒɛərni/), or JavaScript Object Notation, is a text-based open standard designed for human-readable data interchange. It is derived from the JavaScript scripting language for representing simple data structures and associative arrays, called objects. Despite its relationship to JavaScript, it is language-independent, with parsers available for many 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 used primarily to transmit data between a server and web application, serving as an alternative to XML.

History

Douglas Crockford was the first to specify and popularize the JSON format.\(^2\)

JSON was used at State Software, a company co-founded by Crockford, starting around 2001. The JSON.org\(^1\) website was launched in 2002. In December 2005, Yahoo! began offering some of its web services in JSON.\(^3\) Google started offering JSON feeds for its GData web protocol in December 2006.\(^4\)

Although JSON was based on a subset of the JavaScript scripting language (specifically, Standard ECMA-262 3rd Edition—December 1999\(^5\)) and is commonly used with that language, it is a language-independent data format. Code for parsing and generating JSON data is readily available for a large variety of programming languages. JSON's website\(^1\) 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 in JavaScript, generally depends on implementation)
- **String** (double-quoted Unicode, with backslash escaping)
- **Boolean** (true or false)
- **Array** (an ordered sequence of values, comma-separated and enclosed in square brackets; the values do not need to be of the same type)
- **Object** (an unordered collection of key:value pairs with the ':' character separating the key and the value, comma-separated and enclosed in curly braces; the keys must be strings and should be distinct from each other)
- **null** (empty)

Non-significant white space may be added freely around the "structural characters" (i.e. the brackets "[[]]", colon ":" and comma ",").

<table>
<thead>
<tr>
<th>Filename extension</th>
<th>.json</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet media type</td>
<td>application/json</td>
</tr>
<tr>
<td>Uniform Type Identifier</td>
<td>public.json</td>
</tr>
<tr>
<td>Type of format</td>
<td>Data interchange</td>
</tr>
<tr>
<td>Extended from</td>
<td>JavaScript</td>
</tr>
<tr>
<td>Standard(s)</td>
<td>RFC 4627</td>
</tr>
<tr>
<td>Website</td>
<td>json.org (^1)</td>
</tr>
</tbody>
</table>
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.

```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"
        }
    ]
}
```

Since JSON is a subset of JavaScript, it is possible (but not recommended, because it allows the execution of arbitrary JavaScript wrapped in function blocks) to parse 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` as follows:

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

The `contact` variable must be wrapped in parentheses to avoid an ambiguity in JavaScript’s syntax.[6]

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 being executed inadvertently.

Browsers, such as Firefox 4 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.[7]
Unsupported native data types

JavaScript syntax defines several native data types not included in the JSON standard: Date, Error, Math, Regular Expression, and Function. These JavaScript data types must be represented as some other data format, with the programs on both ends agreeing on how to convert between types. As of 2011, there are some de facto standards for e.g. converting between Date and String, but none universally recognized. Other languages may have a different set of native types that must be serialized carefully to deal with this type of conversion.

Schema

There are several ways to verify the structure and data types inside a JSON object, much like an XML schema; however unlike XML schema, JSON schemas are not widely used. Additionally JSON Schema have to be written manually; unlike XML, there are currently no tools available to generate a JSON schema from JSON data.

JSON Schema 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 the 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 was written up as an IETF draft, which expired in 2011. But there are several validators currently available for different programming languages, each with varying levels of conformance. Currently the most complete and compliant JSON Schema validator available is JSV.

Example JSON Schema:

```json
{
   "name": "Product",
   "properties": {
      "id": {
         "type": "number",
         "description": "Product identifier",
         "required": true
      },
      "name": {
         "type": "string",
         "description": "Name of the product",
         "required": true
      },
      "price": {
         "type": "number",
         "minimum": 0,
         "required": true
      },
      "tags": {
```
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"],
    "stock": { "warehouse":300, "retail":20 }
}
```

**MIME type**

The official MIME type for JSON text is "application/json".[15]

**Use in Ajax**

JSON is often used in Ajax techniques. Ajax is a term for the ability of a webpage to request new data after it has loaded into the web browser, usually in response to user actions on the displayed webpage. As part of the Ajax model, the new data is usually incorporated into the user interface display dynamically as it arrives back from the server. An example of this in practice might be that while the user is typing into a search box, client-side code sends what they have typed so far to a server that responds with a list of possible complete search terms from its database. These may be displayed in a drop-down list beneath the search, so that the user may stop typing and select a complete and commonly used search string directly. When it was originally described in the mid-2000s, Ajax commonly used XML as the data interchange format but many developers have also used JSON to pass Ajax updates between the server and the client.[16]
The following JavaScript code is one example of a client using XMLHttpRequest to request data in JSON format from a server. (The server-side programming is omitted; it has to be set up to respond to requests at `url` with a JSON-formatted string.)

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

**Security issues**

Although JSON is intended as a data serialization format, its design as a 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 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 most 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. However, there are some Unicode characters that are valid in JSON strings but invalid in JavaScript, so additional escaping would be needed before using a JavaScript interpreter.[17]

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. For example, if the data is itself not trusted, 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):[18]

```javascript
var my_JSON_object = !(/^[\^;:]{0-9.-\-Eaeilnx-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,[19] but this did not occur. It was first added to the Fifth Edition,[20] 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. Not only does this eliminate the eval() security problem above, but also increases performance due to the fact that functions no longer have to be parsed. 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, via JSON.parse() and JSON.stringify():

• Mozilla Firefox 3.5+[21]
• Microsoft Internet Explorer 8+[22]
• Opera 10.5+[23]
• WebKit-based browsers (e.g. Google Chrome, Apple Safari)[24]

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

• YUI Library[25]
• Prototype[26]
• jQuery[27]
• Dojo Toolkit[28]
• MooTools[29]

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[30] module provides support for several forms of referencing including circular, multiple, inter-message, and lazy referencing.[31][32] Alternatively, non-standard solutions exist such as the use of Mozilla JavaScript Sharp Variables, although this functionality has been removed in Firefox version 12.[33]

Comparison with other formats

See also Comparison of data serialization 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.[34] Apart from XML, examples could include OGD, YAML and CSV. Also, Google Protocol Buffers can fill this role, although it is not a data interchange language.

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. When data is encoded in XML, the result is typically larger than an equivalent encoding in JSON, mainly because of XML’s closing tags. Yet, if the data is compressed using an algorithm like gzip there is little difference because compression is good at saving space when a pattern is repeated.

In XML there are alternative ways to encode the same information because some values can be represented both as child nodes and attributes. This can make automated data exchange complicated unless the used XML format is strictly specified as programs need to deal with many different variations of the data structure.

JSON example:

```
{
    "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"
} ]

Both of the following XML examples carry the same information as the JSON example above in different ways.

XML examples:

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

<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>
</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.

References
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. ... So the idea's been around there for a while. What I did was I gave it a specification, and a little website."
[8] RFC 4627
[10] Dates and JSON - Tales from the Evil Empire (http://weblogs.asp.net/bleryo/archive/2008/01/18/dates-and-json.aspx)
External links

- Format home page (http://www.json.org/)
- RFC 4627 - The application/json Media Type for JavaScript Object Notation (JSON)
- Improvements to JSON (http://bolinfest.com/essays/json.html)