# JSON

From Wikipedia, the free encyclopedia
(Redirected from Json)

**JSON** (/ˈdʒeɪsən/ *JAY-sun*, /ˈdʒeɪsɒn/ *JAY-sawn*), 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.

**JSON**

| | |
|---|---|
| **Filename extension** | `.json` |
| **Internet media type** | `application/json` |
| **Uniform Type Identifier** | public.json |
| **Type of format** | Data interchange |
| **Extended from** | JavaScript |
| **Standard(s)** | RFC 4627 |
| **Website** | json.org (http://json.org/) |

## Contents



## 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 (http://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 scripting language (specifically, Standard ECMA-262 3rd Edition —December 1999[4]) 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 (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 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 ",").

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, 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:

```
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 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.[6]

## Unsupported native data types

JavaScript syntax defines several native data types not included in the JSON standard:[7] 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.[8][9] Other languages may have a different set of native types that must be serialized carefully to deal with this type of conversion.

# Mastering JSON ( JavaScript Object Notation )

Filed: Tue, Apr 10 2007 under Programming|| Tags: JSON javascript objects ajax

Widely hailed as the successor to XML in the browser, JSON aspires to be nothing more than a simple, and elegant data form the browser and server; and in doing this simple task it will usher in the next version of the World Wide Web itself.

## The Object: An Introduction

Behold, an Object…

```
var myFirstObject = {};
```

It may not look like much, but those squiggly braces have the potential to record every bit of information humanity has ever g programs computer scientists can dream up. In fact, Javascript itself is stored inside a set of squiggly braces just like that, as a numbers, arrays, dates, regular expressions, they're all objects and they all started out just like myFirstObject.

## Creating A New Object

The old way to create a new object was to use the "new" keyword.

```
var myJSON = new Object();
```

This method has been deprecated now in favor of simply defining an empty object with squigly braces…

```
var myJSON = {};
```

## Objects as Data

At its most base level a Javascript Object is a very flexible and robust data format expressed as "name/value pairs". That is, a property -- think of it as a plain old variable name that's attached to the object name. And the object holds the value of that na

```
var myFirstJSON = { "firstName" : "John",
                    "lastName"  : "Doe",
                    "age"       : 23 };

document.writeln(myFirstJSON.firstName);  // Outputs John
document.writeln(myFirstJSON.lastName);   // Outputs Doe
document.writeln(myFirstJSON.age);        // Outputs 23
```

This object has 3 properties or name/value pairs. The name is a string -- in our example, firstName, lastName, and age. The v remember everything in Javascript is an object so the value can be a string, number, array, function, even other Objects) -- In 23. John and Doe are strings but age is a number and as you can see this is not a problem.

This data format is called JSON for JavaScript Object Notation. What makes it particularly powerful is that since the value ca arrays and other objects, nesting them as deeply as you need. Here is an example of a somewhat complex JSON structure…

```
var employees = { "accounting" : [   // accounting is an array in employees.
                                  { "firstName" : "John",  // First element
                                    "lastName"  : "Doe",
                                    "age"       : 23 },

                                  { "firstName" : "Mary",  // Second Element
                                    "lastName"  : "Smith",
                                    "age"       : 32 }
```

```
                                ], // End "accounting" array.
                    "sales"       : [ // Sales is another array in employees.
                                    { "firstName" : "Sally", // First Element
                                      "lastName"  : "Green",
                                      "age"       : 27 },

                                    { "firstName" : "Jim",   // Second Element
                                      "lastName"  : "Galley",
                                      "age"       : 41 }
                                ] // End "sales" Array.
                } // End Employees
```

Here employees is an object. That object has two properties or name/value pairs. Accounting is an array which holds two JSC employees. Likewise sales is also an array which holds two JSON objects showing the name and ago of the two employees w within the employees object. There are several different ways to access this data.

## Accessing Data In JSON

The most common way to access JSON data is through dot notation. This is simply the object name followed by a period and would like to access.

```
var myObject = { 'color' : 'blue' };

document.writeln(myObject.color); // outputs blue.
```

If your object contains an object then just add another period and name…

```
var myObject = { 'color' : 'blue',
                 'animal' : {'dog' : 'friendly' }
               };

document.writeln(myObject.animal.dog); // outputs friendly
```

Using the "employee" example above, if we wanted to access the first person who worked in sales…

```
document.writeln(employees.sales[0].firstName + ' ' + employees.sales[0].lastName);
```

We can also access the second person who works in "accounting".

```
document.writeln(employees.accounting[1].firstName + ' ' + employees.accounting[1].lastName);
```

To recap, the "employee" example is an object which holds two arrays each of which holds two additional objects. The only l storage and memory available to it. Because JSON can store objects within objects within objects and arrays within arrays tha limit to what a JSON object can store. Given enough memory and storage requirement, a simple JSON data structure can stor ever generated by humanity.

## Simulating An Associative Array

You can also access JSON data as if it were an Associative Array.

```
var myFirstJSON = { "firstName" : "John",
                    "lastName"  : "Doe",
                    "age"       : 23 };

document.writeln(myFirstJSON["firstName"]);  // Outputs John
document.writeln(myFirstJSON["lastName"]);   // Outputs Doe
document.writeln(myFirstJSON["age"]);        // Outputs 23
```

Be aware that this is NOT an associative array, however it appears. If you attempt to loop through myFirstObject you will get any methods or prototypes assigned to the object, so while you're more than free to use this method of addressing JSON data, and not for what it is not (Associative Array).

```
This file was updated October 12, 2013 at 9:00 PM


======================================================================
RichieHavens-VariousAlbums.xml
======================================================================


<?xml version="1.0" encoding="UTF-8"?>
<!--
  Richie Havens: "Mixed Bag"
  Jesse M Heines, heines@cs.uml.edu
  February 12, 2012
  source:  http://musicbrainz.org/ws/2/release/6a866e4d-e2c7-4990-92a5-9d1a30d51943?inc=artist-credit
-->
<metadata xmlns="http://musicbrainz.org/ns/mmd-2.0#">
  <release id="6a866e4d-e2c7-4990-92a5-9d1a30d51943">
    <title>Mixed Bag</title>
    <status>Official</status>
    <quality>normal</quality>
    <text-representation>
      <language>eng</language>
      <script>Latn</script>
    </text-representation>
    <artist-credit>
      <name-credit>
        <artist id="b45ea9f4-d0a1-45e1-b3a8-bf4fe509b8d0">
          <name>Richie Havens</name>
          <sort-name>Havens, Richie</sort-name>
        </artist>
      </name-credit>
    </artist-credit>
    <date>1967</date>
    <country>US</country>
    <asin>B000001FOM</asin>
    <label-info-list count="1">
      <label-info>
        <catalog-number>FTS-3006</catalog-number>
        <label id="87607d99-1448-4c5f-b794-b67671862aa0">
          <name>Verve Forecast</name>
          <sort-name>Verve Forecast</sort-name>
        </label>
      </label-info>
    </label-info-list>
    <medium-list count="1">
      <medium>
        <position>1</position>
        <format>Vinyl</format>
        <disc-list count="0" />
        <track-list count="11" offset="0">
          <track>
            <position>1</position>
            <length>217866</length>
            <recording id="5b076ec1-01c6-4924-aa54-4a22322d5561">
              <title>High Flyin' Bird</title>
              <length>217866</length>
              <artist-credit>
                <name-credit>
                  <artist id="b45ea9f4-d0a1-45e1-b3a8-bf4fe509b8d0">
                    <name>Richie Havens</name>
                    <sort-name>Havens, Richie</sort-name>
                  </artist>
                </name-credit>
              </artist-credit>
            </recording>
          </track>
```

```
 ... additional tracks and albums go here ...

 70         </track-list>
 71       </medium>
 72     </medium-list>
 73   </release>
 74 </metadata>
 75
 76
 77 ========================================================================
 78 RichieHavens-VariousAlbums.json
 79 ========================================================================
 80
 81
 82 {
 83   "release" : {
 84     "id" : "6a866e4d-e2c7-4990-92a5-9d1a30d51943",
 85     "title" : "Mixed Bag" ,
 86     "status" : "Official" ,
 87     "quality" : "normal" ,
 88     "text-representation" : {
 89       "language" : "eng" ,
 90       "script" : "Latn"
 91     } ,
 92     "artist-credit" : {
 93       "name-credit" : {
 94         "artist" : {
 95           "id" : "b45ea9f4-d0a1-45e1-b3a8-bf4fe509b8d0" ,
 96           "name" : "Richie Havens" ,
 97           "sort-name" : "Havens, Richie"
 98         }
 99       }
100     } ,
101     "date" : "1967" ,
102     "country" : "US" ,
103     "asin" : "B000001FOM" ,
104     "label-info-list" : {
105       "count=" : "1" ,
106       "label-info" : {
107         "catalog-number" : "FTS-3006" ,
108         "label" : {
109           "id" : "87607d99-1448-4c5f-b794-b67671862aa0" ,
110           "name" : "Verve Forecast" ,
111           "sort-name" : "Verve Forecast"
112         }
113       }
114     } ,
115     "medium-list" : {
116       "count" : "1" ,
117       "medium" : {
118         "position" : "1" ,
119         "format" : "Vinyl" ,
120         "disc-list" : {
121           "count" : "0"
122         } ,
123         "track-list" : {
124           "count" : "11" ,
125           "offset" : "0" ,
126           "track" : {
127             "position" : "1" ,
128             "length" : "217866" ,
129             "recording" : {
130               "id" : "5b076ec1-01c6-4924-aa54-4a22322d5561" ,
131               "title" : "High Flyin' Bird" ,
132               "length" : "217866" ,
133               "artist-credit" : {
134                 "name-credit" : {
```

```
135                     "artist" : {
136                       "id" : "b45ea9f4-d0a1-45e1-b3a8-bf4fe509b8d0" ,
137                       "name" : "Richie Havens" ,
138                       "sort-name" : "Havens, Richie"
139                     }
140                   }
141                 }
142               }
143             }
144
```

... additional tracks and albums go here ...

```
145           }
146         }
147       }
148     }
149   }
150
151
152   ========================================================================
```