Json

SON: **J**ava**S**cript **O**bject **N**otation.

JSON is syntax for storing and exchanging text information. Much like XML.

JSON is smaller than XML, and faster and easier to parse.

JSON Example

{  
"employees": [  
{ "firstName":"John" , "lastName":"Doe" },   
{ "firstName":"Anna" , "lastName":"Smith" },   
{ "firstName":"Peter" , "lastName":"Jones" }  
]  
}

The employees object is an array of 3 employee records (objects).

What is JSON?

* JSON stands for **J**ava**S**cript **O**bject **N**otation
* JSON is lightweight text-data interchange format
* JSON is language independent **\***
* JSON is "self-describing" and easy to understand

|  |  |
| --- | --- |
| **\*** | JSON uses JavaScript syntax for describing data objects, but JSON is still language and platform independent. JSON parsers and JSON libraries exists for many different programming languages. |

JSON - Evaluates to JavaScript Objects

The JSON text format is syntactically identical to the code for creating JavaScript objects.

Because of this similarity, instead of using a parser, a JavaScript program can use the built-in eval() function and execute JSON data to produce native JavaScript objects.

Much Like XML

* JSON is plain text
* JSON is "self-describing" (human readable)
* JSON is hierarchical (values within values)
* JSON can be parsed by JavaScript
* JSON data can be transported using AJAX

Much Unlike XML

* No end tag
* Shorter
* Quicker to read and write
* Can be parsed using built-in JavaScript eval()
* Uses arrays
* No reserved words

Why JSON?

For AJAX applications, JSON is faster and easier than XML:

Using XML

* Fetch an XML document
* Use the XML DOM to loop through the document
* Extract values and store in variables

Using JSON

* Fetch a JSON string
* eval() the JSON string

JSON syntax is a subset of JavaScript syntax

JSON Syntax Rules

JSON syntax is a subset of the JavaScript object notation syntax:

* Data is in name/value pairs
* Data is separated by commas
* Curly braces hold objects
* Square brackets hold arrays

JSON Name/Value Pairs

JSON data is written as name/value pairs.

A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

"firstName" : "John"

This is simple to understand, and equals to the JavaScript statement:

firstName = "John"

JSON Values

JSON values can be:

* A number (integer or floating point)
* A string (in double quotes)
* A Boolean (true or false)
* An array (in square brackets)
* An object (in curly brackets)
* null

JSON Objects

JSON objects are written inside curly brackets,

Objects can contain multiple name/values pairs:

{ "firstName":"John" , "lastName":"Doe" }

This is also simple to understand, and equals to the JavaScript statements:

firstName = "John"  
lastName = "Doe"

JSON Arrays

JSON arrays are written inside square brackets.

An array can contain multiple objects:

{  
"employees": [  
{ "firstName":"John" , "lastName":"Doe" },   
{ "firstName":"Anna" , "lastName":"Smith" },   
{ "firstName":"Peter" , "lastName":"Jones" }  
]  
}

In the example above, the object "employees" is an array containing three objects. Each object is a record of a person (with a first name and a last name).

JSON Uses JavaScript Syntax

Because JSON uses JavaScript syntax, no extra software is needed to work with JSON within JavaScript.

With JavaScript you can create an array of objects and assign data to it like this:

Example

var employees = [  
{ "firstName":"John" , "lastName":"Doe" },   
{ "firstName":"Anna" , "lastName":"Smith" },   
{ "firstName":"Peter" , "lastName": "Jones" }  
];

The first entry in the JavaScript object array can be accessed like this:

employees[0].firstName + " " + employees[0].lastName;

The returned content will be:

John Doe

The data can be modified like this:

employees[0].firstName = "Gilbert";

[**Try it yourself »**](http://www.w3schools.com/json/tryit.asp?filename=tryjson_objectarray)

In the next chapter you will learn how to convert a JSON text to a JavaScript object.

JSON Files

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

**JSON data type**

There are following datatypes supported by JSON format:

|  |  |
| --- | --- |
| **Type** | **Description** |
| Number | double- precision floating-point format in JavaScript |
| String | double-quoted Unicode with backslash escaping |
| Boolean | true or false |
| Array | an ordered sequence of values |
| Value | it can be a string, a number, true or false, null etc |
| Object | an unordered collection of key:value pairs |
| Whitespace | can be used between any pair of tokens |
| null | empty |

## Number

* It is a double precision floating-point format in JavaScript and it depends on implementation.
* Octal and hexadecimal formats are not used.
* No NaN or Infinity is used in Number.

The following table shows number types:

|  |  |
| --- | --- |
| **Type** | **Description** |
| Integer | Digits 1-9, 0 and positive or negative |
| Fraction | Fractions like .3, .9 |
| Exponent | Exponent like e, e+, e-,E, E+, E- |

### SYNTAX:

var json-object-name = { string : number\_value, .......}

### EXAMPLE:

Example showing Number Datatype, value should not be quoted:

var obj = {marks: 97}

## String

* It is a sequence of zero or more double quoted Unicode characters with backslash escaping.
* Character is a single character string i.e. a string with length 1.

The table shows string types:

|  |  |
| --- | --- |
| **Type** | **Description** |
| " | double quotation |
| \ | reverse solidus |
| / | solidus |
| b | backspace |
| f | form feed |
| n | new line |
| r | carriage return |
| t | horizontal tab |
| u | four hexadecimal digits |

### SYNTAX:

var json-object-name = { string : "string value", .......}

### EXAMPLE:

Example showing String Datatype:

var obj = {name: 'Amit'}

## Boolean

It includes true or false values.

### SYNTAX:

var json-object-name = { string : true/false, .......}

### EXAMPLE:

var obj = {name: 'Amit', marks: 97, distinction: true}

## Array

* It is an ordered collection of values.
* These are enclosed square brackets which means that array begins with .[. and ends with .]..
* The values are separated by ,(comma).
* Array indexing can be started at 0 or 1.
* Arrays should be used when the key names are sequential integers.

### SYNTAX:

[ value, .......]

### EXAMPLE:

Example showing array containing multiple objects:

{

"books": [

{ "language":"Java" , "edition":"second" },

{ "language":"C++" , "lastName":"fifth" },

{ "language":"C" , "lastName":"third" }

]

}

## Object

* It is an unordered set of name/value pairs.
* Object are enclosed in curly braces that is it starts with '{' and ends with '}'.
* Each name is followed by ':'(colon) and the name/value pairs are separated by , (comma).
* The keys must be strings and should be different from each other.
* Objects should be used when the key names are arbitrary strings

### SYNTAX:

{ string : value, .......}

### EXAMPLE:

Example showing Object:

{

"id": "011A",

"language": "JAVA",

"price": 500,

}

## Whitespace

It can be inserted between any pair of tokens. It can be added to make code more readable. Example shows declaration with and without whitespace:

### SYNTAX:

{string:" ",....}

### EXAMPLE:

var i= " sachin";

var j = " saurav"

## null

It means empty type.

### SYNTAX:

null

### EXAMPLE:

var i = null;

if(i==1)

{

document.write("<h1>value is 1</h1>");

}

else

{

document.write("<h1>value is null</h1>");

}

## JSON Value

It includes:

* number (integer or floating point)
* string
* boolean
* array
* object
* null

### SYNTAX:

String | Number | Object | Array | TRUE | FALSE | NULL

### EXAMPLE:

var i =1;

var j = "sachin";

var k = null;

# jQuery.parseJSON()

also call JSON.parse();

**Description:***Takes a well-formed JSON string and returns the resulting JavaScript object.*

#### version added: [1.4.1](http://api.jquery.com/category/version/1.4.1/)[jQuery.parseJSON( json )](http://api.jquery.com/jquery.parsejson/#jQuery-parseJSON-json)

* + **json**

Type: [String](http://api.jquery.com/Types/#String)

The JSON string to parse.

Passing in a malformed JSON string results in a JavaScript exception being thrown. For example, the following are all malformed JSON strings:

* {test: 1} (test does not have double quotes around it).
* {'test': 1} ('test' is using single quotes instead of double quotes).

The JSON standard does not permit "control characters" such as a tab or newline. An example like$.parseJSON( '{ "testing":"1\t2\n3" }' ) will throw an error in most implementations because the JavaScript parser converts the string's tab and newline escapes into literal tab and newline; doubling the backslashes like "1\\t2\\n3" yields expected results. This problem is often seen when injecting JSON into a JavaScript file from a server-side language such as PHP.

Where the browser provides a native implementation of JSON.parse, jQuery uses it to parse the string. For details on the JSON format, see <http://json.org/>.

Prior to jQuery 1.9, $.parseJSON returned null instead of throwing an error if it was passed an empty string, null, orundefined, even though those are not valid JSON.

## Example:

#### Parse a JSON string.

|  |  |
| --- | --- |
| 1  2  3 | **var** obj = jQuery.parseJSON( '{ "name": "John" }' );  alert( obj.name === "John" )  ; |

# stringify

[**IN THIS ARTICLE**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify#toc)

* 1. [Summary](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify#Summary)
  2. [Syntax](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify#Syntax)
  3. [Parameters](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify#Parameters)
  4. [Description](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify#Description)
     1. [space argument](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify#space_argument)
     2. [toJSON behavior](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify#toJSON_behavior)

### Summary

Convert a value to JSON, optionally replacing values if a replacer function is specified, or optionally including only the specified properties if a replacer array is specified.

| **Method of**[**JSON**](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/JSON) | |
| --- | --- |
| Implemented in | JavaScript 1.7 |
| ECMAScript Edition | ECMAScript 5th Edition |

### Syntax

JSON.stringify(value[, replacer [, space]])

### Parameters

**value**

The value to convert to a JSON string.

**replacer**

If a function, transforms values and properties encountered while stringifying; if an array, specifies the set of properties included in objects in the final string.  
A detailed description of the replacer function is provided in [Using native JSON#The\_replacer\_parameter](https://developer.mozilla.org/En/Using_native_JSON#The_replacer_parameter).

**space**

Causes the resulting string to be pretty-printed.

### Description

JSON.stringify converts an object to JSON notation representing it.

assert(JSON.stringify({}) === '{}');

assert(JSON.stringify(true) === 'true');

assert(JSON.stringify("foo") === '"foo"');

assert(JSON.stringify([1, "false", false]) === '[1,"false",false]');

assert(JSON.stringify({ x: 5 }) === '{"x":5}');

JSON.stringify({x: 5, y: 6}); // '{"x":5,"y":6}' or '{"y":6,"x":5}'

Properties of non-array objects are not guaranteed to be stringified in any particular order. Do not rely on ordering of properties within the same object within the stringification.

Boolean, Number, and String objects are converted to the corresponding primitive values during stringification, in accord with the traditional conversion semantics.

If undefined, a function, or an XML value is encountered during conversion it is either omitted (when it is found in an object) or censored to null (when it is found in an array).

#### space argument

The space argument may be used to control spacing in the final string. If it is a number, successive levels in the stringification will each be indented by this many space characters (up to 10). If it is a string, successive levels will indented by this string (or the first ten characters of it).

JSON.stringify({ a: 2 }, null, " "); // '{\n "a": 2\n}'

Using a tab character mimics standard pretty-print appearance:

JSON.stringify({ uno: 1, dos : 2 }, null, '\t')

// returns the string:

// '{ \

// "uno": 1, \

// "dos": 2 \

// }'

#### toJSON behavior

If an object being stringified has a property named toJSON whose value is a function, then the toJSON method customizes JSON stringification behavior: instead of the object being serialized, the value returned by the toJSON method when called will be serialized. For example:

var obj = {

foo: 'foo',

toJSON: function () {

return 'bar';

}

};

var json = JSON.stringify({x: obj});

json will be the string '{"x":"bar"}'.

JSON.stringify turns an object in to a JSON text and stores that JSON text in a string.

JSON.parse turns a string of JSON text into an object.