JavaScript JSON

•  [What is JSON?](http://www.dofactory.com/tutorial/javascript-json" \l "section1)

•  [XML versus JSON](http://www.dofactory.com/tutorial/javascript-json" \l "section2)

•  [JSON.parse() and JSON.stringify()](http://www.dofactory.com/tutorial/javascript-json" \l "section3)

•  [Using JSON with AJAX](http://www.dofactory.com/tutorial/javascript-json" \l "section4)

What is JSON?

*JSON* stands for *JavaScript Object Notation*. It is an open, text-based format designed for lightweight data exchange between different systems. JSON is a subset of JavaScript's object literal notation representing simple data structures and objects. The array elements and object member values in JSON are limited to numbers, strings, Boolean values, objects, arrays, and null. This is different from JavaScript object literals where values of array elements and object members can be any JavaScript expression, including function definitions.

The beauty of JSON standard lies in its simplicity and the fact that it is very compact. Let's look at an example. Here we have the JSON representation of a student object.

1. {
2. "firstName": "Kim",
3. "lastName": "Taylor",
4. "studentCode": 7
5. }

JSON objects can be nested. Below is a student including its image, which is an object itself:

1. {
2. "firstName": "Kim",
3. "lastName": "Taylor",
4. "studentCode": 7,
5. "image":
6. {
7. "url": "images/007.jpg",
8. "width": 150,
9. "height": 150
10. }
11. }

A JSON-formatted string always has a single top-level item which is either an object, with curly braces {}, or an array, with square brackets []. The earlier examples are objects. The next example shows an array with three elements:

1. [
2. "Red",
3. "Green",
4. "Blue"
5. ]

The elements in a JSON array can be of different type. Here is an example of an array with four elements, each one of a different data type:

1. [
2. "Blue", // string
3. { // object
4. "id": "001",
5. "name": "IBM"
6. },
7. 2.4, // number
8. True // Boolean
9. ]

JSON doesn't natively represent complex data types like functions, date, and time. To represent date and time values, you can either use an ISO-formatted string or a number (number of milliseconds in UTC since January 1, 1970). Note that these are just conventions: JSON standard does not dictate rules for encoding date and time values.

Although JSON is a subset of the standard JavaScript object notation, it is a language-independent data format. JSON parsers that parse and generate JSON data are readily available for most programming languages. Visit [www.json.org](http://www.dofactory.com/tutorial/www.json.org) and you will find a broad range of available JSON libraries.

JSON maps very well to data structures that most programming languages support. A JSON **object** is an unordered collection of name-value pairs which easily maps to associative arrays (such as Dictionary, Map, and Hash) in other languages. Similarly, a JSON **array** is an ordered list of values which nicely maps to any programming language's array or list types.

Because of these characteristics, JSON is ideal as a data exchange format between disparate applications. There are few if any parsing complexities. It is very easy, for example, to convert a C# object to JSON and then pass it to a Python or JavaScript application.

Today, JSON is widely used for serializing and sending structured data over a network, essentially replacing XML. Most commonly, JSON is used in Ajax communications between a server and the web browser. JSON-formatted data is lightweight and JavaScript can parse it very easily making it ideal for Ajax-style web applications. The data that is transmitted is encoded as JSON objects, and those JSON objects can be transformed with little effort into true JavaScript objects on the browser.

XML versus JSON

Over the last few years, XML has lost ground to JSON as a popular data exchange format. In XML there is no concept of data types. All element values are strings. To add type information, you have to apply the XML Schema typing layer on the top of XML. This adds complexity to your application when you read the values within the XML code. JSON objects provide structure to your data and their values are typed as numbers, strings, Boolean values (true and false), null, nested objects, or arrays. In XML, arrays and objects have to be represented by convention while JSON supports them natively.

JSON also has a clear edge over XML when it comes to parsing data in JavaScript. With XML, data is parsed manually requiring code that transforms parsed text into JavaScript objects. On the other hand, with JSON you can immediately transform JSON-formatted data into JavaScript objects using the JSON.parse() method. Parsing JSON-formatted data is very fast because it is done by the JavaScript engine itself.

While XML parsers are available in all modern browsers, you have to be very careful with cross-browser differences. How you create the DOM that will load the XML depends on which browser you are using. On the other hand, JSON is very well supported by all browsers.

Here is a comparison of a student record formatted as XML and as JSON.

|  |  |
| --- | --- |
| **XML format** | **JSON format** |
| <student> <firstName>Kim</firstName> <lastName>Taylor<lastName> <studentCode>7<studentCode> </student> | {     "firstName": "Kim",      "lastName": "Taylor",     "studentCode" : 7 } |

As you can see, the XML is larger than JSON. XML documents tend to be lengthy because of the presence of both start and end tags and namespaces. JSON, on the other hand, is lightweight and its syntax is rather terse. When sending a large amount of data over a network, JSON is preferred as this will significantly improve performance.

Furthermore, XML has a bit of a learning curve because it is typically used in conjunction with other technologies, including XML Schema, XPath, XSLT, etc. JSON, on the other hand, is easy to learn especially for developers that have a background in JavaScript.

An advantage of XML is that it provides native support for namespaces which reduces the risk of ambiguity when merging documents from different sources. JSON has no notion of namespaces and hence poor extensibility. A possible workaround to prevent naming collisions is to prefix object member names with a context.

JSON.parse() and JSON.stringify()

Instead of employing a third-party JSON parser library, the JavaScript engine can itself be used to process the JSON-formatted data and convert it to native JavaScript objects. As you know, JavaScript has an eval() function that is designed to evaluate a JavaScript expression and return its value. Since JSON is mostly syntactically correct and valid JavaScript code, an effortless approach to execute JSON data would be to use eval().

However, eval() is not recommended because your application would expose itself to all sorts of security vulnerabilities. Even if the source is trusted, a security breach at the trusted source's end could cause a code injection attack in your application. So, eval() should be avoided.

The recommended solution is to use a JavaScript-based JSON parser library in your client code which essentially is a wrapper around eval(). Before the parser evaluates, it validates the input string as JSON and makes sure that it does not contain arbitrary executable JavaScript code thereby preventing potentially malicious code from getting executed.

As a secure alternative to eval(), EcmaScript5 provides the native JSON.parse() which is exclusively meant to parse JSON data and not JavaScript. It is supported by most web modern browsers. Alternatively, you can use json.js, or better, json2.js (downloadable from [https://github.com/douglascrockford/JSON-js](https://github.com/douglascrockford/JSON-j)) which minified is just 2.5K. The JSON.parse() method first validates the JSON-formatted text string and then converts it into a JavaScript object.

Deserializing JSON text into the JavaScript object with JSON.parse()

Here is an example of JSON.parse() in action.

1. var json = '{ "firstName":"Kim", "lastName":"Taylor", "studentCode":7 }';
2. var student = JSON.parse(json);
3. alert(student.firstName + " " + student.lastName); // => Kim Taylor

The optional reviver argument to JSON.parse() is a function that filters and transforms the results . It accepts two arguments, key and value. The reviver is invoked with each of the key-value pairs generated by the parse, and its return value is used instead of the original value. If the reviver returns what it received, the structure is not modified. If the reviver returns null or undefined, the property is deleted from the result. In the example below reviver transforms all date strings to date objects (and transforms these from Greenwich Mean Time (Zulu time) to local time in the browser's time zone.

1. function reviver (key, value) {
2. var num = Date.parse(value);
3. if (num) {
4. return new Date(num);
5. }
6. return value;
7. }
8. var json = '{ "firstName":"Kim", "registrationDate" :
9. "2012-05-20T11:20:00.000Z" }';
10. var student = JSON.parse(json, reviver);
11. alert(student.registrationDate); // => Sun May 20 06:20:00 CDT 2012

Serializing objects into the JSON format with JSON.stringify()

The JSON.stringify() method does the reverse of JSON.parse() and converts a JavaScript object into a JSON-formatted string. The optional replacer argument to JSON.stringify() is a function filters and transform the way a value is stringified. It accepts two arguments, key and value. In the example below, the properties with scores less than 40 are filtered out:

1. var scores = { "Java": 70, "JavaScript": 80, "C#": 30 };
2. var passed = JSON.stringify(scores, function (key, value) {
3. if (value < 40) {
4. return undefined;
5. } else {
6. return value;
7. }
8. });
9. alert(passed); // {"Java":70,"JavaScript":80}

[Run](javascript:void(0);)

Using JSON with AJAX

JSON is frequently used with Ajax. The example below demonstrates a common situation of converting a JavaScript object to JSON, using JSON.stringify(), and then sending it to the server using Ajax. It also demonstrates the reverse of receiving a JSON-formatted response from the server and converting it into a JavaScript object by using JSON.parse().

1. // Get the browser specific XmlHttpRequest
2. function createXHRObject() {
3. if (window.ActiveXObject) { // IE
4. return new ActiveXObject("Microsoft.XMLHTTP");
5. } else if (window.XMLHttpRequest) {
6. return new XMLHttpRequest();
7. }
8. }
9. function getStudentScore() {
10. var student = new Student("Kim","Taylor", "JavaScript Training");
11. var json = JSON.stringify(student);
12. var xmlHttp = createXHRObject();
13. // open the connection to the web server with the
14. // XMLHttpRequest object. score.php is the web server's
15. // page from where we will get the JSON response
16. xmlHttp.open("POST", "score.php", true);
17. // stores the name of a function to be called
18. // automatically each time the readyState property changes
19. xmlHttp.onreadystatechange = handleServerResponse;
20. // send the header information along with the request
21. xmlHttp.setRequestHeader("Content-type",
22. "application/x-www-form-urlencoded");
23. xmlHttp.setRequestHeader("Content-length", json.length);
24. xmlHttp.setRequestHeader("Connection", "close");
26. // send the JSON-encoded Student object to the server
27. // using the XMLHttpRequest object
28. xmlHttp.send(json);
29. }
30. function handleServerResponse() {
31. // readyState holds the status of the XMLHttpRequest.
32. // 4 means the request is finished and response is ready.
33. if (xmlHttp.readyState == 4) {
34. if (xmlHttp.status == 200) { // 200 indicates okay
35. // Deserializing JSON text into the JavaScript object
36. var studentScore = JSON.parse(xmlHttp.responseText);
37. // do something with it ...
38. }
39. } else if (xmlHttp.status == 400) { // 400 indicates file not found
40. //report error
41. }
42. }

As far as the Ajax part is concerned most web sites will use a library such as jQuery and let it handle browser differences. Using jQuery significantly reduces the size of the code as demonstrated below:

1. function getStudentScore() {
2. var student = new Student("Kim","Taylor", "JavaScript Training");
3. var json = JSON.stringify(student);
4. $.post("score.php", json, function (data) {
5. var studentScore = JSON.parse(data);
6. // do something with it ...
7. }, "json" );
8. }