Mastering JSON (JavaScript Object Notation)

By Patrick Hunlock

Widely hailed as the successor to XML in the browser, JSON aspires to be nothing more than a simple, and elegant data format for the exchange of information between 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 gathered, and express the most complex programs computer scientists can dream up. In fact, Javascript itself is stored inside a set of squiggly braces just like that, as are all of its primitive data types -- strings, 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, an object holds a name which is an object's 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 name. Here's an example…

```
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 value can be **any** Javascript object (and remember everything in Javascript is an object so the value can be a string, number, array, function, even other Objects) -- In this example our values are John, Doe, and 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 can be any data type, you can store other 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 JSON objects showing the names and age of 2 employees. Likewise sales is also an array which holds two JSON objects showing the name and ago of the two employees who work in sales. All of this data exists 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 then followed by the name/property you 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 limits to the structure are the amount of storage and memory available to it. Because JSON can store objects within objects within objects and arrays within arrays that can also store objects, there is no virtual limit to what a JSON object can store. Given enough memory and storage requirement, a simple JSON data structure can store, and properly index, all the information 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, in addition to the three properties above, any methods or prototypes assigned to the object, so while you're more than free to use this method of addressing JSON data, just treat it for what it is (Object Properties) and not for what it is not (Associative Array).

Receiving JSON via AJAX

There are three separate ways to receive JSON data via AJAX. Assignment, Callback, and Parse.

JSON Via Assignment

There's no standard naming convention for these methods, however assignment method is a good descriptive name because the file coming in from the server creates a Javascript expression which will assign the JSON object to a variable. When the `responseText` from the server is passed through `eval`, `someVar` will be loaded with the JSON object and you can access it from there.

```
// example of what is received from the server.
var JSONFile = "someVar = { 'color' : 'blue' }";

eval(JSONFile); // Execute the javascript code contained in JSONFile.
document.writeln(someVar.color); // Outputs 'blue'
```

JSON Via Callback

The second method calls a pre-defined function and passes the JSON data to that function as the first argument. A good name for this method is the callback method. This approach is used extensively when dealing with third party JSON files (IE, JSON data from domains you do not control).

```
function processData(incommingJSON) {
   document.writeln(incommingJSON.color); // Outputs 'blue'
}
// example of what is received from the server...
var JSONFile = "processData( { 'color' : 'blue' } )";
eval(JSONFile);
```

JSON Via Parse

The third and final method sends a raw object which must be parsed by a function. This could be referred to as the parse method. This is, by far, the safest and most secure way to transfer JSON data and it will be a part of the next version of Javascript due to be released in 2008. For now, it is, unfortunately, limited only to domains which you control.

```
// The following block implements the string.parseJSON method
(function (s) {
  // This prototype has been released into the Public Domain, 2007-03-20
  // Original Authorship: Douglas Crockford
  // Originating Website: http://www.JSON.org
  // Originating URL    : http://www.JSON.org/JSON.js
  // Augment String.prototype. We do this in an immediate anonymous function to
  // avoid defining global variables.
  // m is a table of character substitutions.
  var m = {
    '\b': '\\b',
    '\t': '\\t',
    '\n': '\\n',
    '\f': '\\f',
    '\r': '\\r',
    '"' : '\\"',
    '\\': '\\\\'
  };
  s.parseJSON = function (filter) {
    // Parsing happens in three stages. In the first stage, we run the text against
    // a regular expression which looks for non-JSON characters. We are especially
    // concerned with '()' and 'new' because they can cause invocation, and '='
    // because it can cause mutation. But just to be safe, we will reject all
    // unexpected characters.
    try {
      if (/^("(\\.|[^"\\\n\r])*?"|[,:{}\[\]0-9.\-+Eaeflnr-u \n\r\t])+?$/.
        test(this)) {
          // In the second stage we use the eval function to compile the text into a
          // JavaScript structure. The '{' operator is subject to a syntactic ambiguity
          // in JavaScript: it can begin a block or an object literal. We wrap the text
          // in parens to eliminate the ambiguity.
          var j = eval('(' + this + ')');
          // In the optional third stage, we recursively walk the new structure, passing
          // each name/value pair to a filter function for possible transformation.
          if (typeof filter === 'function') {
            function walk(k, v) {
              if (v && typeof v === 'object') {
                for (var i in v) {
                  if (v.hasOwnProperty(i)) {
                    v[i] = walk(i, v[i]);
                  }
                }
              }
              return filter(k, v);
            }
            j = walk('', j);
          }
          return j;
        }
      } catch (e) {
      // Fall through if the regexp test fails.
      }
      throw new SyntaxError("parseJSON");
    };
  }
) (String.prototype);
// End public domain parseJSON block
// begin sample code (still public domain tho)
JSONData = '{"color" : "green"}';   // Example of what is received from the server.
testObject=JSONData.parseJSON();
document.writeln(testObject.color); // Outputs: Green.
```

As you can see, you'll need to include the public domain prototype which will parse the JSON data, however once it's included, processing JSON data is as simple as it looks in the last three lines of the above example. Of the three extraction methods, the parse method is the most secure and exposes your code to the fewest problems. You should use this method wherever possible in all of your JSON requests via AJAX.

Retrieving JSON Data Via Ajax

For these examples we'll be using the Ajax framework from the article: The Ultimate Ajax Object.

When you are communicating with your own servers, AJAX is one of the better ways to transfer data from the server to the browser. Provided the server is under your control you can be reasonably assured this method of data transfer is safe. Here is an example of a simple AJAX request which communicates with the server, retrieves some data, and passes it along to a processing function. We'll be using the callback method here where the JSON object will execute a pre-defined function after it's loaded, in this case -- `processData(JSONData);`

```
function processData(JSONData) {
   alert(JSONData.color);
}
var ajaxRequest = new ajaxObject('http://www.somedomain.com/getdata.php');
    ajaxRequest.callback = function (responseText) {
       eval(responseText);
    }
    ajaxRequest.update();

// In this example we assume the server sends back the following data file
// (which the ajax routine places in responseText)
//
// processData( { "color" : "green" } )
```

In this example our data file, because it's actually Javascript code, when passed through the eval statement will execute `processData`, passing our actual JSON data to the function as the first argument.

The next example will use the parse method and assumes you have the `parseJSON` prototype located elsewhere in your code.

```
function processData(JSONData) {
   alert(JSONData.color);
}
var ajaxRequest = new ajaxObject('http://www.somedomain.com/getdata.php');
    ajaxRequest.callback = function (responseText) {
       JSONData = responseText.parseJSON();
       processData(JSONData);
    }
    ajaxRequest.update();
// In this example we assume the server sends back the following data file
// (which the ajax routine places in responseText)
//
// { "color" : "green" }
```

Now when the the server returns the JSON file, it's parsed on the line which reads `JSONData = responseText.parseJSON();` and then `JSONData` is passed to `processData`. The end result is the same as the first example, but if, for some reason, the JSON data contained malicious or invalid data then the second example would be more likely to securely handle any problems.

Howto Send JSON Data To The Server

Because AJAX data is sent as an encoded string, some preparation of JSON data must be made before it can be sent to the server. Fortunately, Douglas Crockford at JSON.org has released a set of very useful routines which will convert any Javascript data type into a JSON string which can be easily sent to the server.

The source for this library can be obtained at http://www.JSON.org/JSON.js. The code is public domain and is as easy to use as clipping the data and pasting it into your toolbox.

The following example defines a JSON object then uses the `toJSONString()` method to convert the object into a string which is ready to be sent to the server.

```
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
var toServer = employees.toJSONString();
document.writeln(toServer);
```

This outputs:

```
//Outputs:
//{"accounting":[{"firstName":"John","lastName":"Doe","age":23},{"firstName":"Mary
","lastName":"Smith","age":32}],"sales":[{"firstName":"Sally","lastName":"Green","
age":27},{"firstName":"Jim","lastName":"Galley","age":41}]}
```

Retreiving JSON From Third Party Servers

This section comes with a bunch of warnings. Up until this point, JSON and AJAX has been relatively secure since you are communicating with servers under your control, receiving data that is under your control. With third party services all of this changes.

As of Internet Explorer 7 and Firefox 2, use of third party JSON data exposes your web page to malicious attacks and great security risks. The moment you request third party data you have given up control of your web page and the third party can do whatever they want with it from scraping the data and sending back sensitive information to installing listeners to wait for sensitive data to be entered.

For most trusted sources the only thing an external JSON request will get you is a bunch of useful data. However if the page making the third party request contains form elements, requires a secure/encrypted connection, or contains ANY personal or confidential data you absolutely, positively should NOT use third-party JSON calls on that page.

Before you attempt to use third party JSON data take a good, hard look at the page you are creating and now imagine that the worst sort of person is looking at that page with you. If you are uncomfortable with what that other person can do with the information displayed on that page, you should not be using third-party JSON calls.

Quite a few services are starting to offer JSON in addition to RSS/XML as data formats. Yahoo, in particular, is quite progressive at implementing JSON. What's very cool about JSON is that the web page can directly request and process that data, unlike XML which must be requested by the server and then passed along to the browser. Unfortunately there is no RSS/FEED standard for JSON, though most services try to emulate RSS XML in a JSON structure.

For the following example we'll use this site's JSON feed, a simple one-to-one conversion of XML to JSON. The feed is even generated by the exact same program which generates the XML version, only the arguments passed on the URL determine which format is sent. The URL for this site's feed is:

```
http://www.hunlock.com/feed.php?format=JSON&callback=JSONFeed
```

All third party JSON requests use the callback method where, once the JSON file has been loaded it will call a specific function, passing the JSON data as the first argument. Most services have a default function name they will call, and some services will allow you to change the name of the function which is called. My JSON feed will call `JSONFeed()` by default but you can change this by changing the name in the URL. If you changed the URL to read…

```
http://www.hunlock.com/feed.php?format=JSON&callback=processJSON
```

…now when the feed has been loaded it will call the function named `processJSON` and pass the JSON data as the first argument. This is fairly important when you are handling multiple JSON requests on your page.

All third party JSON requests are loaded via the `<script>` tag, just like you would use to load external Javascript files. What's a little different is that we create the script tag manually and attach it to the page manually as well. This is a fairly simple process however and the code to do it is quite simple and readable. Here's a small function which will accept a URL and load it.

```
function loadJSON(url) {
  var headID = document.getElementsByTagName("head")[0];
  var newScript = document.createElement('script');
      newScript.type = 'text/javascript';
      newScript.src = url;
  headID.appendChild(newScript);
}
```

This function gets the (first) `<head>` element of our page, creates a new script element, gives it a type, sets the `src` attribute to the URL which is passed and then appends the new script element to the page `<head>` Once appended, the script will be loaded and then executed.

Here is a very simple little program to get this site's JSON feed and display the items.

```
function loadJSON(url) {
  var headID = document.getElementsByTagName("head")[0];
  var newScript = document.createElement('script');
      newScript.type = 'text/javascript';
      newScript.src = url;
  headID.appendChild(newScript);
}
function processJSON(feed){
  document.writeln(feed.title+'<BR>');
  for(i=0; i<feed.channel.items.length; i++) {
     document.write("<a href='"+feed.channel.items[i].link+"'>");
     document.writeln(feed.channel.items[i].title+"</a><BR>");
  }
}
loadJSON('http://www.hunlock.com/feed.php?format=JSON&callback=processJSON');
```

This code creates a function to load third party Javascript (`loadJSON`) and a function to process the data it receives (`processJSON`). The last line calls `loadJSON` and passes it the URL to use. Once the script has finished loading its executed automatically which will call `processJSON` (because that's the name we specified in the callback), and `processJSON` will loop through all the items in the file showing the article title as a clickable link.

Yahoo Pipes

Almost all of the web is still hidden behind XML (RSS) which means you need a server to actually do anything at all with that data. With Yahoo Pipes however you can transform any RSS/XML feed into JSON. To do this, simply find the URL of the RSS feed you would like to use and append it to the end of the following line…

```
http://pipes.yahoo.com/pipes/9oyONQzA2xGOkM4FqGIyXQ/run?&_render=JSON&_callback=piper&feed=
```

This is a yahoo pipe which will accept a RSS/XML feed and convert it to JSON. Using this little tool your web pages can DIRECTLY deal with ANY XML/RSS data on the web without the need for a server side script.

Using our example above, modified now to read dzone's rss feed:

```
function loadJSON(url) {
  var yahooPipe =
'http://pipes.yahoo.com/pipes/9oyONQzA2xGOkM4FqGIyXQ/run?&_render=JSON&_callback=processJSON&feed=';
  var headID = document.getElementsByTagName("head")[0];
  var newScript = document.createElement('script');
      newScript.type = 'text/javascript';
      newScript.src = yahooPipe+url;
  headID.appendChild(newScript);
}
function processJSON(feed){
  document.writeln(feed.value.title+'<BR>');
  for(i=0; i<feed.value.items.length; i++) {
     document.write("<a href='"+feed.value.items[i].link+"'>");
     document.writeln(feed.value.items[i].title+"</a><BR>");
  }
}
loadJSON('http://feeds.dzone.com/dzone/frontpage');
```

For more information on how to use yahoo pipes, this see the article: Yahoo Pipes--RSS without Server Side Scripts.

JSON Security

Most of the hysteria over JSON security involves very far-fetched scenarios. One scenario, for instance, assumes a user can be "tricked" to visiting a web page and that web page initiates a third party `<script>` JSON-request to a secure site and sniffs for sensitive data it gets back. This attack attempts to trick the server into thinking the user is on the server's site and requesting JSON data. It assumes the victim (server) is serving JSON as a callback or assignment -- if the server is sending a raw JSON file meant to be parsed, this attack is ineffective because a `<script>` can't initiate the parse, only AJAX can parse raw objects sent in `responseText` and AJAX can not be used across different domains.

This is the reason this article (and most security professionals) recommend you serve sensitive JSON data as a raw JSON file which must be passed through a `parseJSON` method.

And of course, it never hurts to mention again that if your web pages contain ANY sensitive data what-so-ever, they should never attempt to use `<script>` tags to load third-party JSON data.

A fairly decent overview of JSON security vulnerabilities can be found here: (http://www.fortifysoftware.com/servlet/downloads/public/JavaScript_Hijacking.pdf)

An overview of VERY exotic AJAX security vulnerabilities via prototype attacks can be found here: Subverting_Ajax.pdf http://events.ccc.de/congress/2006/Fahrplan/attachments/1158-Subverting_Ajax.pdf

JSON Best Practices

In the wake of the above security report, Douglas Crockford (Senior JavaScript Architect at Yahoo Inc) wrote a blog entry which provided very simple and elegant methods to avoid most every JSON security problem. Those methods boiled down to…

- Never Trust The Browser
- Keep Data Clean (Don't embed functions in your JSON data)
- Avoid Third Party JSON
- Authenticate on the server side all JSON data requests (make sure they're on your site)
- Use SSL (Browser Encryption) for sensitive data.

The future of JSON

Right now JSON is in its infancy, however it's a pretty big baby. PHP supports JSON in version 5. And the first cracks in XML's domination of RSS are just starting to appear.

In 2008 most browsers will be able to parse JSON and to convert any variable into JSON. There is also an upcoming standard (also from Douglas Crockford) which will enable a browser to securely request a JSON file from outside the current domain. When both of these technologies have been implemented, we will be entering the world of Web 3.0 where any browser can request any JSON published data anywhere in the world and manipulate it to its own ends without the need to involve any proxy servers.

XML when it was introduced made waves, but JSON, when it is fully supported by the browser, will make tsunamis and if you're not already preparing for that day, and the new applications browser-supported JSON will enable then you're very much cheating yourself out of a front-row seat at the next revolution.

Version

This document is current as of Firefox 2.03, IE 7.0, JSCRIPT 5.6, ECMA-262 Edition 3, Javascript 1.7

License