IBM

developerWorks®

# Get to know JsonML

## Use JsonML to build and extend UI elements

Martin Brown

03 July 2007

The rise of JavaScript Object Notation (JSON) has gone hand-in-hand with the rise of Asynchronous JavaScript + XML (Ajax). JSON is useful because it enables you to easily transmit data that can be turned back into a JavaScript object, but it still requires custom scripting to deal with that object. JsonML is an extension of JSON that enables you to map XML data using JSON type markup, and this in turn enables you to easily create XML or XHTML data based on JSON markup and to build and exchange user interface (UI) elements. This article shows you how to make use of this handy tool.

## Exchanging data in Ajax

One of the most common problems with the Ajax interface is that the exchange of information between the client browser and the host server normally needs some kind of encoding and post-processing or parsing to convert the data stream into something that is usable directly within the JavaScript application.

Within Ajax that method of encapsulation is XML, a worldwide and well used and understood standard that also has some problems and limitations. The main problem with XML is that the encoding process is expensive and that, on most browsers, the decoding of the XML response is equally time consuming.

Frequently with an Ajax application, the request is based on the contents of the form. The response will be a representation of the information returned by the server in a series of Java objects that you can use to display the information. The resulting sequence is similar to that shown in Figure 1.

**Figure 1. Typical Ajax process**

The process in Figure 1 has two problems:

- The first is that working with XML is generally very complex. Whether you generate XML in the first place or consume XML from the server, the process is time consuming and tedious, and frequently inefficient.
- The other problem is that the conversion of the data into the display format is complex. Either you need to build the UI elements by hand using the DOM model or supply the data in preformatted XHTML to begin with. The latter method forces the server to provide the UI elements (which might in itself be a dangerous proposition). The former just adds overhead to a process that you want to work as efficiently as possible.

The DOM model is also affected by the fact that different Web browsers and environments have different DOM structures and interfaces. This just adds further complexity to the display of the information you load, and it's complexity you don't want or need.

A number of solutions to the problem are available. The first part of the solution is to change how data is exchanged from the XML format to JSON. That's not the focus of this article, but understanding JSON will make JsonML easier to understand. JsonML provides the other half of the solution—making the actual display of the information much easier to produce. Let's look at JSON first.

## Understanding JSON

JavaScript Object Notation (JSON) aims to solve most of the issues with the use of XML, the parsing of that content, and it's translation into a more usable internal structure by utilizing the abilities of the JavaScript standard to exchange information.

At it's heart JSON is still a text format, but it is more human readable and is more compatible with how you generally create nested data objects in many languages (including Perl, PHP, Python, Java, Ruby) and that works well with the JavaScript object format (which is essentially just a nested data structure anyway).

For example, you might model a list of businesses in XML like that shown in Listing 1.

## Listing 1. XML address data

```
<business>
<name>One on Wharf</name>
<address>
    <street>1 Wharf Road</street>
    <town>Grantham</town>
    <county>Lincolnshire</county>
</address>
<phonenumbers>
    <phone>01476 123456</phone>
    <phone>01476 654321</phone>
</phonenumbers>
</business>
```

To parse this within JavaScript requires use of the DOM model to access the components. For example, to get the phone numbers from this structure within JavaScript you might use code like that shown in Listing 2.

## Listing 2. Web application that consumes XML data

```
<html>
<head>
<script type="text/javascript">
var xmlDoc;
function loadXML()
{
// code for IE
if (window.ActiveXObject)
  {
  xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
  xmlDoc.async=false;
  xmlDoc.load("/json/address.xml");
  getaddress();
  }
// code for Mozilla, Firefox, Opera, etc.
else if (document.implementation &&
document.implementation.createDocument)
  {
  xmlDoc=document.implementation.createDocument("","",null);
  xmlDoc.load("/json/address.xml");
  xmlDoc.onload=getaddress;
  }
else
  {
  alert('Your browser cannot handle this script');
  }
}
function getaddress()
{
document.getElementById("business").innerHTML=
xmlDoc.getElementsByTagName("name")[0].childNodes[0].nodeValue;
document.getElementById("phone").innerHTML=
xmlDoc.getElementsByTagName("phone")[0].childNodes[0].nodeValue;
document.getElementById("fax").innerHTML=
xmlDoc.getElementsByTagName("phone")[1].childNodes[0].nodeValue;
}
</script>
```

```
</head>
<body onload="loadXML()">
<h1>Address</h1>
<p><b>Business</b> <span id="business"></span><br />
<b>Phone:</b> <span id="phone"></span><br />
<b>Fax:</b> <span id="fax"></span>
</p>
</body>
</html>
```

The code in Listing 2 gets the information of just one address, accessing elements of the original XML document, such as the name of the business using this logic: `xmlDoc.getElementsByTagName("name")[0].childNodes[0].nodeValue`.

This is quite ugly!

Take a look at the same information in JSON, shown here in Listing 3.

## Listing 3. JSON version of address data

```
{
    "business" : {
        "name" : "One on Wharf",
        "address" : {
            "street" : "1 Wharf Road",
            "town" : "Grantham",
            "county" : "Lincolnshire",
        },
        "phonenumbers" : [
                          "01476 123456",
                          "01476 654321",
                          ]
                      }
}
```

Notice that the structure is much simpler and easier to read. The markup format used in JSON can be evaluated directly by JavaScript without the need for a parser using `eval()`, and the result is just another JavaScript object: `var addressbookObj = eval('(' + jsontext + ')');`.

**Note:** Ideally you shouldn't use `eval()`, since it can be used to execute arbitrary text. A number of JSON parsers that will only parse JSON text are available.

Once the information is an object within JavaScript then you can use standard JavaScript notation to access the information. For example, you can get the same name and phone numbers from a JSON data source that has been parsed into an object (see Listing 4).

## Listing 4. JSON data source parsed into an object

```
addressbookObj.business.name
addressbookObj.business.phonenumbers[0]
addressbookObj.business.phonenumbers[1]
```

Listing 4 is much simpler and easier to use and access. You can write a complete page that uses the above data structure and provides the same information as your original XML example, shown here in Listing 5.

### Listing 5. Web application that consumes JSON data

```
<html>
<head>
<script type="text/javascript" src="prototype.js"></script>
<script type="text/javascript">

function showaddress(req)
{
    var addressbookObj = eval('(' + req.responseText + ')');

    document.getElementById("business").innerHTML =
       addressbookObj.business.name;
    document.getElementById("phone").innerHTML =
       addressbookObj.business.phonenumbers[0];
    document.getElementById("fax").innerHTML =
       addressbookObj.business.phonenumbers[1];
}

function getaddress()
{
   new Ajax.Request('address.json',
                    { method : 'get',
                      onSuccess : showaddress,
                      onFailure: function(){
              alert('Something went wrong...') }});
}
</script>
</head>
<body onload="getaddress()">
<h1>Address</h1>
<p><b>Business:</b> <span id="business"></span><br />
<b>Phone:</b> <span id="phone"></span><br />
<b>Fax:</b> <span id="fax"></span>
</p>
</body>
</html>
```

I used the Prototype library here to load the original JSON file, but you can see that the process of actually parsing and then displaying the information in the JSON file is much simpler. However, you still have to employ the DOM to actually populate the output with the information that was loaded from the JSON file.

Returning to the original JSON, the JSON is comparatively smaller in terms of the data transferred, and in a larger Ajax project the payload overhead of XML can be considerable, especially if you merge that with the reduced JavaScript overhead to actually process the content.

The above example shown in Listing 5 is a simple demonstration of the simplicity of the JSON standard. The key to most Ajax applications is actually the customized display of information. In this example, you've used the DOM model to embed the contents of the XML or JSON data into the HTML page. In many Ajax applications, however, you might have more complex user interface elements to build.

## JsonML

JSON Markup Language uses the basic mechanics of the JSON data interchange format and applies it to the representation of XML so that you can exchange XML-formatted data using a similar simplified text-based markup.

This sounds backwards—a notation format that enables you to model XML data as an alternative to using XML directly—but consider the structure and readability of JSON compared to the original XML. With JsonML you can take advantage of many of the same advantages: readability and data sizes are two obvious advantages.

The main focus of JsonML is as a tool for building UI elements. Traditionally you have two ways to develop this kind of browser-based user interface:

- Generate XHTML on the server and insert into the current page using the `innerHTML` attribute of a DOM object.
- Build the DOM structure by hand using the client DOM interface.

This is, as you've seen, a somewhat cumbersome method, particularly with large datasets where the repetitive nature of the formatting can serve to confuse the process. Plus, you still have the problem of dealing with browser-sensitive DOM implementations.

This is what JsonML aims to resolve. It melds the simplicity of the JSON markup with the DOM as the target format. You get the same benefits; the source is easily loadable and parseable by a JavaScript client (without having to worry about the DOM implementation in use). Also, because you can describe everything in the JsonML document, you have effectively embedded both the data and the markup into a single file. Because you don't need the DOM to build the output, you don't need a complex parsing process to generate the UI elements.

If you remember, back in Figure 1, the typical method to display information recovered through an Ajax interface involves requesting the data, parsing the XML, converting the XML to the DOM structure required by the browser and then actually displaying the output.

Using JsonML, you can replace the whole XML to XHTML through the DOM and JavaScript stages with a single stage.

## JsonML markup

Before you look at some JsonML examples, it is worthwhile to look at the structure of JsonML. The official format of the structure is shown in Listing 6.

### Listing 6. JsonML markup format

```
element
    = '[' tag-name ',' attributes ',' element-list ']'
    | '[' tag-name ',' attributes ']'
    | '[' tag-name ',' element-list ']'
    | '[' tag-name ']'
    | json-string
    ;
tag-name
    = json-string
    ;
attributes
    = '{' attribute-list '}'
    | '{' '}'
    ;
attribute-list
    = attribute ',' attribute-list
    | attribute
    ;
attribute
    = attribute-name ':' attribute-value
    ;
attribute-name
    = json-string
    ;
attribute-value
    = json-string
    ;
element-list
    = element ',' element-list
    | element
    ;
```

Although this looks complicated, the basic structure is very easy to follow.

For a start, any element should be text, so you can create an un-numbered list element like this: `[ "ul" ]`.

The square brackets group the element logically. If you want to add attributes to the element, use braces as the next element in the list within the group. For example, to change the style so that there are no bullets, you would use: `[ "ul", { style : "list-style-type:none"} ]`.

Of course, a list is no use without any children; they are added as further elements to the same list (see Listing 7).

### Listing 7. List with children

```
[ "ul", { style : "list-style-type:none"},
    [ "li",
      "First item"],
    [ "li",
      "Second item"],
    [ "li",
      "Third item"],
    ];
```

The attributes to an item are optional, so you can omit them, as shown in Listing 7 with the individual list items.

## Parsing JsonML to XHTML

The above examples are JSON, so you can define them within a JavaScript application natively. To parse the content into an object that actually displays as XHTML, you need the JsonML library from the JsonML Web site. To parse a JSON object from JsonML into XHTML you use the `parseJsonML()` method. Once the JsonML is parsed, it is immediately accessible as a UI element that you can add to your page.

For example, Listing 8 shows a complete Web page that will add your bulleted list to a predefined container each time you click on the supplied link.

### Listing 8. Web application that displays a JsonML template

```
<html>
<head>
```

```
<script type="text/javascript" src="JsonML.js"></script>
<script type="text/javascript">

var listblock = [ "ul",
                  [ "li",
                    "First item"],
                  [ "li",
                    "Second item"],
                  [ "li",
                    "Third item"],
                  ];

function addblock (id,jsonmlblock)
{
    var aselement = listblock.parseJsonML();
    var container = document.getElementById(id);
    container.appendChild(aselement);
}

</script>
</head>
<body>
<a href"#list" onclick="addblock('container',
 listblock);return false;">Add block</a>
<div id="container"/>
</body>
</html>
```

Figure 2 shows the Web page in action, shown here after the link was clicked a couple of times with the bulleted list imported into the page.

### Figure 2. Simple JsonML example in action

Let's go through the example step by step. The first block is the JsonML that represents your list item (see Listing 9).

### Listing 9. JsonML representing the list item

```
var listblock = [ "ul",
                  [ "li",
                    "First item"],
                  [ "li",
                    "Second item"],
                  [ "li",
                    "Third item"],
                  ];
```

It is just a JavaScript object. When the user clicks on the link, the `addblock()` function is called. This has just three steps. First, parse the JavaScript object into XHTML by applying the `parseJsonML()` method: `var aselement = jsonmlblock.parseJsonML();`.

Now you get the destination container: `var container = document.getElementById(id);`.

And, finally, you add the generated XHTML and append it to the container: `container.appendChild(aselement);`.

## Parsing existing XHTML or XML into JsonML

Before you start to build applications that use JsonML, you might have existing XML or XHTML structures that you want to convert into JsonML. The JsonML site (see Resources) provides an XSL transform that will output JsonML from an XML source.

To use it, download the JsonML.xsl transform and then use xsltproc to perform the translation. For example, take the XML structure in Listing 10.

### Listing 10. XML structure

```
<table class="maintable">
<tr class="odd">
<th>Situation</th>
<th>Result</th>
</tr>
<tr class="even">
<td><a href="driving.html" title="Driving">Driving</a></td>
<td>Busy</td>
</tr>
...
</table>
```

You could translate Listing 10 to JsonML like this: `$ xsltproc JsonML.xsl.xml samp.xml`.

The resulting output is shown in Listing 11. Note that this has been formatted, since the XSL removes new lines from the output for brevity, although you'll notice that they are retained within the actual output as string elements.

### Listing 11. Resulting output

```
["table",
 {"class":"maintable"},
 "\n",
 ["tr",
  {"class":"odd"},
  "\n",
  ["th",
   "Situation"],
  "\n",
  ["th",
   "Result"],
  "\n"],
 "\n",
 ["tr",
  {"class":"even"},
  "\n",
  ["td",
   ["a",
    {"href":"driving.html",
     "title":"Driving"},
    "Driving"]],
  "\n",
  ["td",
   "Busy"],
  "\n"],
 ]
```

You can use this method to convert any structure you want from XML into JsonML.

## Parsing existing DOM structures into JsonML

Another situation you might have is that you have built an XML or XHTML block and you need to convert it to JsonML. A JavaScript function is available that will perform the conversion for you. This is available from the JsonML Web site.

## Using JsonML as a UI builder

You can use JsonML as a UI builder in a number of ways. One easy way is to create a JavaScript structure in JsonML that acts as the UI you want to build. Once you build the JavaScript structure, convert the structure in a single action into XHTML and insert the generated XHTML into the page. For example, Listing 12 shows a UI builder for table cells that enables you to add a black or white cell to a row of a table, add rows to the table and, ultimately, output the table, it's JsonML source and the corresponding XHTML source.

### Listing 12. JsonML-based table builder

```
<html>
<head>
<script type="text/javascript" src="JsonML.js"></script>
<script type="text/javascript">

var blackcell = [ "td", {"style" : "background-color:black"},"CELL"];
var whitecell = [ "td", {"style" : "background-color:white"},"CELL"];

var currentrow = new Array();
var currenttable = new Array();

function onLoad()
{
    initrow();
    inittable();
}

function initrow()
{
    currentrow = new Array();
    currentrow.push("tr");
}

function inittable()
{
    currenttable = new Array();
    currenttable.push("table");
    currenttable.push({"border" : "1"});
}

function addrow()
{
    currenttable.push(currentrow);
    currentrow = new Array();
    currentrow.push("tr");
    showsource();
}

function addcell(color)
{
    if (color == "black")
        {
            currentrow.push(blackcell);
        }
    else
        {
            currentrow.push(whitecell);
        }
}

function showsource()
{
    var tablelement = currenttable.parseJsonML();
    var container = document.getElementById("viewabletable");
    container.removeChild(container.lastChild);
    container.appendChild(tablelement);

    var srccontainer = document.getElementById("sourceoutput");
    srccontainer.value = currenttable.toSource();

    var domcontainer = document.getElementById("domsourceoutput");
    domcontainer.value = container.innerHTML;
 }

function showtable()
{
    showsource();
    initrow();
    inittable();
}

</script>
</head>
<body onload="onLoad()">
<a href"#addrow" onclick="addrow();return false;">Add Row</a><br/>
<a href"#addbcell" onclick="addcell('black');return false;"
  >Add Black Cell</a><br/>
<a href"#addwcell" onclick="addcell('white');return false;"
  >Add White Cell</a><br/>
<a href"#showtable" onclick="showtable();return false;"
  >Show Table</a><br/>
<a href"#showsource" onclick="showsource();return false;"
  >Show Source</a><br/>

<div id="viewabletable">
</div>
<b>JsonML Source</b>
<textarea rows="20" cols="120" id="sourceoutput"></textarea><br/>
```

```
<b>DOM Source</b>
<textarea rows="20" cols="120" id="domsourceoutput"></textarea><br/>

</body>
</html>
```

The operation of the application is quite simple. Each time you click on the Add Black/White Cell button, you add an array populated with the right information (in this case, a table cell with the right text and formatting) in JsonML format to the array that makes up the current row. When you click on Add Row, you add the array that relates to the row to the array that relates to the entire table. In each case, you only extend the definition of the internal object in it's JsonML notation.

When you click Show Table the internal JavaScript structure is parsed into the XHTML to display the table using the `parseJsonML()` method to convert the original JsonML into a rendered format. You also display the JsonML source (by using the `toSource()` method on the JavaScript object), and the XHTML source by dumping the generated XHTML source. Because you only deal with internal JavaScript structures, you never have to touch the DOM interface to create the HTML, except to actually insert the XHTML into the final document.

Note that although you built the table using static text elements, you might alter the table data before it is inserted into the row. JsonML templates are just JavaScript objects, so you can update the content using a simple assignment: `blackcell[2] = "Some other content";`.

Because the content of the template is so easily updated, you eliminated one part of the complex UI building process: how to convert the information you load through an Ajax connection into the XHTML used for display.

## Binding Behaviors to a JsonML element

The above example (Listing 12) provides a very simple method to generate a table with an internal structure that can be used to generate the equivalent XHTML. But what if you want to change the output to match a particular style, or want to change the given template to match the style and output guidelines that are applicable within the current page?

One way to use JsonML is to generate UI templates that you import into the page at the time of loading (see Listing 13).

### Listing 13. JsonML generates UI templates at the time of loading

```
<script type="text/javascript" src="resultstable.js"></script>
<script type="text/javascript" src="resultline.js"></script>
<script type="text/javascript" src="resultselectionpopup.js"></script>
```

Now to output the different elements of the page, you load each object, convert it to XHTML with `parseJsonML`, and output the result. CSS can only go so far though, and the mechanism shown in Listing 13 provides absolutely no way to introduce any type of information or data into the output.

In a traditional Ajax/DOM model you are able to make these decisions because you parse each element individually and can choose when and where to apply different elements. You cannot interact with the JsonML parsing process in the same way, but you can add a callback binding to the parsing stage that parses each element and outputs the results.

The binding function is executed for each XHTML tag that is generated from the JsonML source, and because the item is a standard HTML object, you can use any comparison or determination method you want to alter the output of the template.

For example, Listing 14 shows another JsonML formatted table.

### Listing 14. JsonML template for a table

```
var table = ["table",{ "border" : "1" },
             ["tr",
              ["td",{"class" : "highlight-effect"},"Highlighted row"],
              ],
             ["tr",
              ["td",{"class" : "non-highlight"},"Non-Highlighted row"],
              ],
             ];
```

When parsed it generates a simple table. You can use the CSS classes to provide the formatting you want. However, you might not always want to highlight the row, only when the data that you insert into the table contains something important.

To achieve this, you can write a binding function that will alter elements with the highlight-effect class to have a different background color, as shown in Listing 15.

### Listing 15. Adding a binding function to a JsonML template

```
function actionbindings(elem) {
    if (elem.className.indexOf("highlight-effect") >= 0) {
        elem.style.backgroundColor = "red";
    }
    return elem;
}
```

To apply the binding, supply the binding function to the `parseJsonML()` function: `var renderedtable = table.parseJsonML(actionbindings);`.

The binding function has full access to the element as formatted in the original JsonML, so the binding can check and change the format based on many attributes, including the tagname, classname and id. All you have to do is modify the element before returning it to the parser that will then insert the altered element into the DOM tree.

## JsonML and XML

Although this article concentrates on the use of JsonML as a solution for XHTML, you can use JsonML for nearly any XML data, since XHTML is a subset of the main XML standard.

This facility can be useful if you want to store static XML fragments in your JavaScript applications, but have the ability to exchange the document into XML when the applications exchange information with other clients.

In particular, because JsonML is accessible as a structure within JavaScript you can easily update the content of the XML structure, then convert to XML and send it to the server.

For example, you can take a simple XML structure as shown in Listing 16.

### Listing 16. Simple XML structure

```
<name>
<firstname>Martin</firstname>
<lastname>Brown</lastname>
</name>
```

In JsonML, this is created as shown in Listing 17.

### Listing 17. Simple JsonML structure

```
["name",
  ["firstname", "Martin"],
  ["lastname", "Brown"],
]
```

As a JavaScript object you can then update the details, shown in Listing 18.

### Listing 18. Update with JavaScript

```
address[1][1] = "Sharon";
```

And then generate an XML version of the object using `parseJsonML()` to create what is shown in Listing 19.

### Listing 19. parseJsonML generates the XML version of the object

```
<name>
<firstname>Sharon</firstname>
<lastname>Brown</lastname>
</name>
```

The above process is much simpler than generating an XML structure using the DOM parser, or by using text manipulation (which is open to abuse and mistakes). The basic templates in each case can be small and instantly accessible.

## Summary

Working with Ajax applications introduces a number of problems, including how to effectively exchange data and then how to format that data for display in your application. By definition, the information in an Ajax application is dynamic and that means building the XHTML used to format that information. Writing XHTML by hand is time consuming and using the DOM model is fraught with problems since the DOM interface differs between browser implementations.

JsonML builds on the basics of JSON and enables you to model the user interface elements using JavaScript notation. This makes the elements easier to build and easier to populate than directly generating XHTML- or DOM-based elements, and it much more compatible. You can then convert the JavaScript structure into XHTML without having to worry about the DOM interface or its differences.

This article covered the fundamentals of the JSON and JsonML standards and how easy they are to update. It also covered a number of examples that show how to use JsonML to build UI elements. You also saw how to extend the JsonML content by processing during the parsing stage, further extending and enhancing the method of outputting XHTML.