

Cross-domain communications with JSONP, Part 1: Combine JSONP and jQuery to quickly build powerful mashups

Skill Level: Intermediate

[Seda Özses](mailto:seda.ozses@at.ibm.com) (seda.ozses@at.ibm.com)

IT Specialist

IBM

[Salih Ergül](#)

IT Architect

Independent Consultant

24 Feb 2009

With the number of publicly offered Web service APIs, it's now much easier to get content from different Web sources and to build mashups—if you have access to the right APIs and tools. Discover how you can combine an obscure cross-domain call technique (JSONP) and a flexible JavaScript library (jQuery) to build powerful mashups surprisingly quickly.

Introduction

Asynchronous JavaScript and XML (Ajax) is the key technology driving the new generation of Web sites, popularly termed as Web 2.0 sites. Ajax allows for data retrieval in the background without interfering with the display and behavior of the Web application. Data is retrieved using the `XMLHttpRequest` function, which is an API that lets client-side JavaScript make HTTP connections to remote servers. Ajax is also the driving force behind many mashups, which integrate content from multiple sources into a single Web application.

This approach, however, does not allow cross-domain communication because of restrictions imposed by the browser. If you try to request data from a different domain, you will get a security error. You can stop these security errors if you control

the remote server where data resides and every request goes to the same domain, but what's the fun of a Web application if you are stuck on your own server? What if you need to collect data from multiple third-party servers?

Understanding the same-origin policy limitations

The same-origin policy prevents a script loaded from one domain from getting or manipulating properties of a document from another domain. That is, the domain of the requested URL must be the same as the domain of the current Web page. This basically means that the browser isolates content from different origins to guard them against manipulation. This browser policy is quite old and dates back to Netscape Navigator 2.0.

One relatively simple way to overcome this limitation is to have the Web page request data from the Web server it originates from, and to have the Web server behave as a proxy relaying the request to the actual third-party servers. Although widely used, this technique isn't scalable. Another way is to use frame elements to create new areas in the current Web page, and to fetch any third-party content using GET requests. After being fetched, however, the content in the frames would be subject to the same-origin policy limitations.

A more promising way to overcome this limitation is to insert a dynamic script element in the Web page, one whose source is pointing to the service URL in the other domain and gets the data in the script itself. When the script loads, it executes. It works because the same-origin policy doesn't prevent dynamic script insertions and treats the scripts as if they were loaded from the domain that provided the Web page. But if this script tries to load a document from yet another domain, it will fail. Fortunately, you can improve this technique by adding JavaScript Object Notation (JSON) to the mix.

JSON and JSONP

JSON is a lightweight data format (compared to XML) for the exchange of information between the browser and server. JSON's appeal to JavaScript developers comes from the fact that it's a string representation of a JavaScript object (hence the name). For example, assume you have a ticker object with two attributes: symbol and price. This is how you can define the ticker object in JavaScript:

```
var ticker = {symbol: 'IBM', price: 91.42};
```

And this is its JSON representation:

```
{symbol: 'IBM', price: 91.42}
```

Refer to [Resources](#) for more information about JSON and its potential uses as a data interchange format. Listing 1 defines a JavaScript function that shows IBM's share price when called. (We are leaving out the exact details of how you can incorporate this into a Web page.)

Listing 1. Defining a showPrice function

```
function showPrice(data) {  
    alert("Symbol: " + data.symbol + ", Price: " + data.price);  
}
```

You can call this function by passing JSON data as a parameter:

```
showPrice({symbol: 'IBM', price: 91.42}); // alerts: Symbol: IBM, Price: 91.42
```

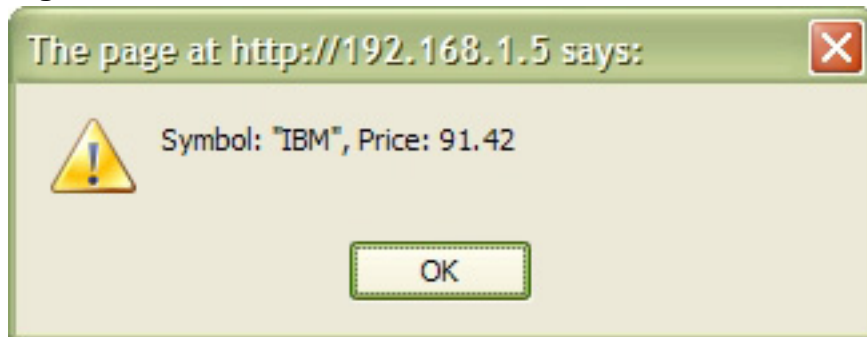
Now you're ready to include these two steps into a Web page, as shown in Listing 2.

Listing 2. Including the showPrice function and parameters in a Web page

```
<script type="text/javascript">  
function showPrice(data) {  
    alert("Symbol: " + data.symbol + ", Price: " + data.price);  
}  
</script>  
<script type="text/javascript">showPrice({symbol: 'IBM', price: 91.42});</script>
```

After loading the page, you should see the alert shown in Figure 1.

Figure 1. IBM ticker



This article, until now, has showed how you can call a JavaScript function with static JSON data as a parameter. However, by wrapping your JSON data dynamically in a function call, you could call your function with the dynamic data, which is a technique called dynamic JavaScript insertion. To see how this work, put the following line in a stand-alone JavaScript file called ticker.js.

```
showPrice({symbol: 'IBM', price: 91.42});
```

Now change the script in your Web page to look like the code shown in Listing 3.

Listing 3. Dynamic JavaScript insertion code

```
<script type="text/javascript">
// This is our function to be called with JSON data
function showPrice(data) {
    alert("Symbol: " + data.symbol + ", Price: " + data.price);
}
var url = "ticker.js"; // URL of the external script
// this shows dynamic script insertion
var script = document.createElement('script');
script.setAttribute('src', url);

// load the script
document.getElementsByTagName('head')[0].appendChild(script);
</script>
```

In the example in Listing 3, the dynamically inserted JavaScript code, residing in the file `ticker.js`, calls the `showPrice()` function using the actual JSON data as a parameter.

As you have already learned, the same-origin policy doesn't prevent the insertion of dynamic script elements into the document. That is, you could dynamically insert JavaScript from different domains, carrying JSON data in them. This is actually what JSONP (JSON with Padding) is: JSON data wrapped in a function call. Note that, in order to do this, you must have a callback function already defined in the Web page at the time of insertion, which is `showPrice()` in our example.

What we call a JSONP service (or a Remote JSON Service), however, is a Web service with the additional capability of supporting the wrapping of the returned JSON data in a user-specified function call. This approach relies on the remote service accepting a callback function name as a request parameter. It then generates a call to this function, passing the JSON data as parameter, which upon arrival at the client is inserted into the Web page and executed.

jQuery's JSONP support

Beginning with version 1.2, jQuery has had native support for JSONP calls. You can load JSON data located on another domain if you specify a JSONP callback, which can be done using the following syntax: `url?callback=?`.

jQuery automatically replaces the `?` with a generated function name to call. Listing 4 shows this code.

Listing 4. Using the JSONP callback

```
jQuery.getJSON(url+"&callback=?", function(data) {
    alert("Symbol: " + data.symbol + ", Price: " + data.price);
});
```

```
});
```

To do this, jQuery attaches a global function to the window object that is called when the script is inserted. This function is removed upon completion. Furthermore, jQuery has an optimization for non-cross-domain calls as well. If the request is being made to the same domain, then jQuery turns it into an ordinary Ajax request.

Example service with JSONP support

In the previous example, you used a static file (`ticker.js`) to dynamically insert JavaScript into a Web page. Although it returns a JSONP reply, it doesn't let you define a callback function name in the URL. It is not a JSONP service. So, how can you transform it to a real JSONP service? Well, there are a variety of ways, and we're going to show you two examples, using PHP and Java.

First, assume that your service accepts a parameter called `callback` in the request URL. (The parameter name is not crucial, but both the client and server must agree on the name.) Also assume that a request to the service looks like this:

```
http://www.yourdomain.com/jsonp/ticker?symbol=IBM&callback=showPrice
```

`symbol`, in this case, is a request parameter representing the requested ticker symbol, and `callback` is the name of your callback function in your Web application. You could call this service with jQuery's JSONP support using the code shown in Listing 5.

Listing 5. Calling the callback service

```
jQuery.getJSON("http://www.yourdomain.com/jsonp/ticker?symbol=IBM&callback=?",
function(data) {
    alert("Symbol: " + data.symbol + ", Price: " + data.price);
});
```

Note that we put a `?` as the callback function name instead of a real function name. This is because jQuery replaces the `?` with a generated function name (like `jsonp1232617941775`) that calls the inline function. This frees you from defining functions like `showPrice()`.

Listing 6 shows an extract from a JSONP service implemented in PHP.

Listing 6. Extract from JSONP service in PHP

```
$jsonData = getDataAsJson($_GET['symbol']);
echo $_GET['callback'] . '(' . $jsonData . ');';
// prints: jsonp1232617941775({"symbol" : "IBM", "price" : "91.42"});
```

Listing 7 shows a Java™ Servlet method doing the same function.

Listing 7. JSONP service in a Java servlet

```
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
    String jsonData = getDataAsJson(req.getParameter("symbol"));
    String output = req.getParameter("callback") + "(" + jsonData + "));";

    resp.setContentType("text/javascript");

    PrintWriter out = resp.getWriter();
    out.println(output);
    // prints: jsonp1232617941775({"symbol" : "IBM", "price" : "91.42"});
}
```

So, what if you want to build mashups, integrating content from third-party servers with the intent of presenting them in a single Web page? The answer is simple: You must use third-party JSONP services, and there are quite a few of them.

Ready-made JSONP services

Now that you know how to use JSONP, you can start using some ready-made JSONP Web services to build your applications and mashups. Following are some starting points for your next development projects. (Hint: You may copy-and-paste the given URLs into the address field of your browser to examine the resulting JSONP response.)

Digg API: Top stories from Digg:

```
http://services.digg.com/stories/top?apikey=http%3A%2F%2Fmashup.com&type=javascript
&callback=?
```

Geonames API: Location info for a zip-code:

```
http://www.geonames.org/postalCodeLookupJSON?postalcode=10504&country=US&callback=?
```

Flickr API: Most recent cat pictures from Flickr:

```
http://api.flickr.com/services/feeds/photos_public.gne?tags=cat&tagmode=any
&format=json&jsoncallback=?
```

Yahoo Local Search API: Search pizza in zip-code location 10504:

```
http://local.yahooapis.com/LocalSearchService/V3/localSearch?appid=YahooDemo&query=pizza
&zip=10504&results=2&output=json&callback=?
```

A cautionary note

JSONP is a very powerful technique for building mashups, but, unfortunately, it is not a cure-all for all of your cross-domain communication needs. It has some drawbacks that must be taken into serious consideration before committing development resources. First and foremost, there is no error handling for JSONP calls. If the dynamic script insertion works, you get called; if not, nothing happens. It just fails silently. For example, you are not able to catch a 404 error from the server. Nor can you cancel or restart the request. You can, however, timeout after waiting a reasonable amount of time. (Future jQuery versions may have an abort feature for JSONP requests.)

Another major drawback of JSONP is that it can be quite dangerous if used with untrusted services. Because a JSONP service returns a JSON response wrapped in a function call, which will be executed by the browser, this makes the hosting Web application vulnerable to a variety of attacks. If you are going to use JSONP services, it's very important to be aware of the threats it poses. (See [Resources](#) for more information.)

Conclusion

In this first article of this series, we explained how to combine JSONP with jQuery to quickly build powerful mashups. We explained the following topics:

- Limitations of the browser same-origin policy and how these are overcome
- JSONP as an effective cross-domain communication technique, by-passing the same-origin policy limitations
- The potential of JSONP to enable Web application developers to build mashups quickly
- A sample JSONP service and its usage: the Ticker service

The next article in the series will present the Yahoo Query Language (YQL) as a single endpoint JSONP service that lets you query, filter, and combine data across the Web. We will also build a sample mashup application using YQL and jQuery.

Resources

- Discover [JSON](#), JavaScript Object Notation APIs for the Java language and other languages.
- Read "[Mastering Ajax, Part 10: Using JSON for data transfer](#)" (developerWorks, March 2007) and learn about JSON as a data interchange format for Ajax applications.
- Get to know jQuery in the three-part article series "[Working with jQuery](#)" (developerWorks, September 2008).
- The [Browser Security Handbook](#) provides a detailed description of the same-origin policy.
- "[Overcome security threats for Ajax applications](#)" (developerWorks, June 2007) gives tips and best practices to secure your mashup applications.
- "[Shaping the future of secure Ajax mashups](#)" (developerWorks, April 2007) explains how to improve the browser for hybrid Web applications.
- Browse the [technology bookstore](#) for books on these and other technical topics.

About the authors

Seda Özses

Seda Özses is a member of the ibm.com corporate webmaster team. Her main focus is on the ibm.com corporate Web portals, where she manages the XSL work and the requirements for her team. You can contact her at seda.ozses@at.ibm.com.

Salih Ergül

Salih Ergül is a self-employed IT architect who specializes in large, mission-critical Enterprise Application Integration projects in the telecom and banking industries. He has authored several articles on Java programming and currently works in the banking industry.

Trademarks

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the

United States, other countries, or both.