JSONP

For security reasons, Ajax requests are limited to the same domain. That is, the page making an Ajax request must be on the same server as the page responding to the request. This is a policy enforced by the web browser to keep one site from maliciously attempting to contact another site (like your bank!). There’s one way around that however. While a web browser can’t send an AJAX XMLHTTP request to a different website, it can download resources from other sites, including pictures, style sheets, and external JavaScript files.

JSONP (which stands for JSON with padding) provides one way to retrieve information from another site. Basically, instead of making an Ajax request of the foreign site, you load a script that contains the JSON code in it. In other words, it’s like linking to an external JavaScript file on Google.

**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.

A 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 for the exchange of information between the browser and server. 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}

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 (exercise 1.html)**

<script>

function showPrice(data) {

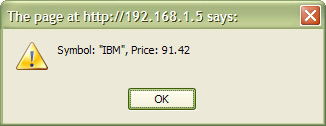
alert("Symbol: " + data.symbol + ", Price: " + data.price);

}

</script>

<script>showPrice({symbol: 'IBM', price: 91.42});</script>

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



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.

Now change the script in your Web page to look like the code shown in Listing 3. (exercise 2.html)

**Listing 3. Dynamic JavaScript insertion code**

<script>

// 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);

});

**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']);

// database acces

echo $\_GET['callback'] . '(' . $jsonData . ');';

// prints: jsonp1232617941775({"symbol" : "IBM", "price" : "91.42"});

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

**Listing 7. JSONP service in Java**

@Override

protected void doGet(HttpServletRequest req, HttpServletResponse resp)

throws ServletException, IOException {

String jsonData = getDataAsJson(req.getParameter("symbol"));

//database

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. (Hint: You may copy-and-paste the given URLs into the address field of your browser to examine the resulting JSONP response.)

Geonames API: Location info for a zip-code:

http://www.geonames.org/postalCodeLookupJSON?postalcode=10118&country=US&callback=myCallback

Flickr API: Most recent cat pictures from Flickr:

http://api.flickr.com/services/feeds/photos\_public.gne?tags=cat&tagmode=any

&format=json&jsoncallback=showCAt

Example: <https://www.autoaddress.ie/support/developer-centre/api/postcode-lookup> is this JSON

Try <https://api.autoaddress.ie/2.0/PostcodeLookup?callback=myCallback>