

Cross-domain communications with JSONP, Part 2: Building mashups with JSONP, jQuery, and Yahoo! Query Language

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

03 Mar 2009

In the [previous](#) article of this series, we introduced JSONP (JSON with Padding) as a way to overcome browser same-origin policy limitations while combining and presenting data from third-party sources. This article continues this process and shows you how to use Yahoo! Query Language (YQL), a JSONP service from Yahoo!, to build a mashup Web page using jQuery.

Introduction

In Part 1 of this series, we introduced JSONP as an effective cross-domain communication technique, one that lets you bypass the same-origin policy limitations imposed by the current browsers. We showed how you can use jQuery's native JSONP support to gather JSON-formatted content from third-party services, which we call JSONP services. This article introduces Yahoo! Query Language (YQL) as a single endpoint JSONP service that lets you query, filter, and combine data from different data sources.

In addition, this article shows you how to build a jQuery implementation of a mashup application that collects stock quotes, *The New York Times* RSS feed, and weather forecast data to present in a single Web page.

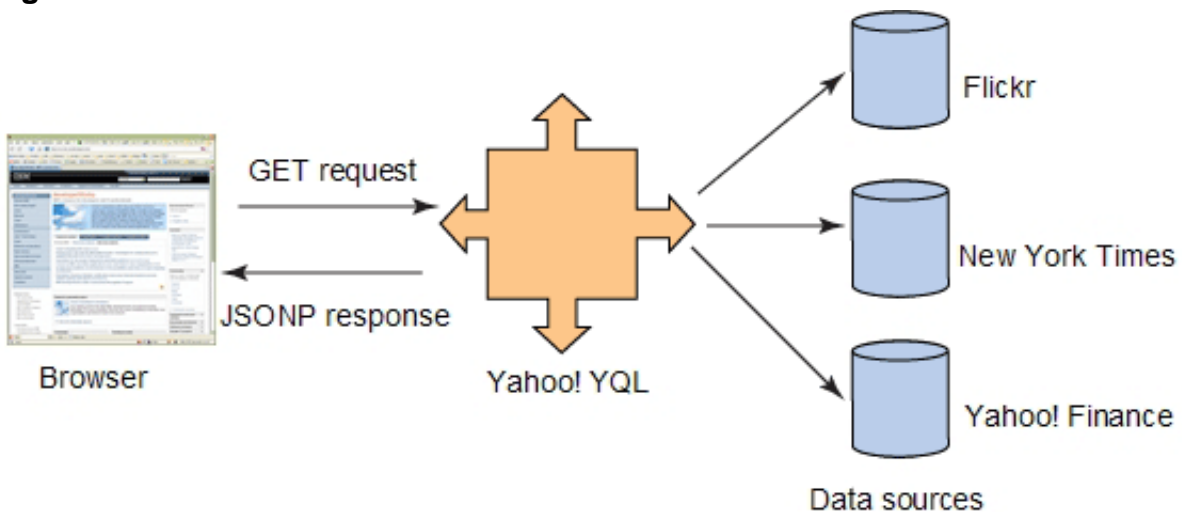
Web as a database

There is a lot of structured data on the Web that is exposed to developers through various Web services or APIs. These services require you to access and query them, and then parse results in non-standard ways. Each one of them has its own URL for access as well as documents describing how to query and understand the results. You can view YQL as a unified way to query data served up by different services at different URLs. YQL provides a single endpoint service that lets you query, filter, and combine data across the Web without having to deal with each service's peculiarities.

YQL is basically a query language based on SQL that lets you query and retrieve structured data from multiple services in a standard way. Yet YQL is also a query engine, which is hosted by Yahoo!, that is exposed at a REST endpoint. YQL uses the familiar relational database model of tables and rows. However, data is interpreted and structured as XML internally. If the underlying data source does not provide XML data, YQL does the conversion. It can return responses as either XML or JSON (which you can specify while calling the service). Moreover, you can also specify a callback function if you request JSON data from the service. This makes the YQL service a JSONP service, which we think is very significant: It brings automatic JSONP support to many Web services.

YQL understands and supports data sources like RSS, Atom, JSON, XML, CSV, HTML, Flickr, Yahoo! Finance, Weather, and so on. Because YQL also supports external data sources (external to Yahoo!), the possibilities of building mashup applications are endless. We could say that it connects the Web to your applications in a single interface.

Figure 1. YQL as a mediator service



As the article already mentioned, YQL is a SQL-like language to query structured data in the Web. Hence, `SELECT` is the main verb in YQL statements. The syntax is

also very similar to SQL:

```
SELECT fields FROM source_table WHERE filter ..
```

The following listing shows a simple query to retrieve the headlines from *The New York Times* RSS feed:

```
select title from rss where url="http://www.nytimes.com/services/xml/rss/nyt/HomePage.xml"
```

A more involved example of a YQL query to retrieve vegetarian restaurants in New York, limiting the number to 3, would be:

```
select Title, Address, City, Rating.AverageRating from local.search
where query="vegetarian" and location="New York, NY" limit 3
```

The above query would return a JSON response like what is shown in Listing 1 (omitting returned metadata about the query).

Listing 1. JSON query response

```
"results": {
  "Result": [
    {
      "Title": "World of Vegetarian Incorporated",
      "Address": "24 Pell St",
      "City": "New York",
      "Rating": {
        "AverageRating": "5"
      }
    },
    {
      "Title": "Counter",
      "Address": "105 1st Ave",
      "City": "New York",
      "Rating": {
        "AverageRating": "4"
      }
    },
    {
      "Title": "Red Bamboo",
      "Address": "140 W 4th St",
      "City": "New York",
      "Rating": {
        "AverageRating": "4.5"
      }
    }
  ]
}
```

You can even query the Web through YQL:

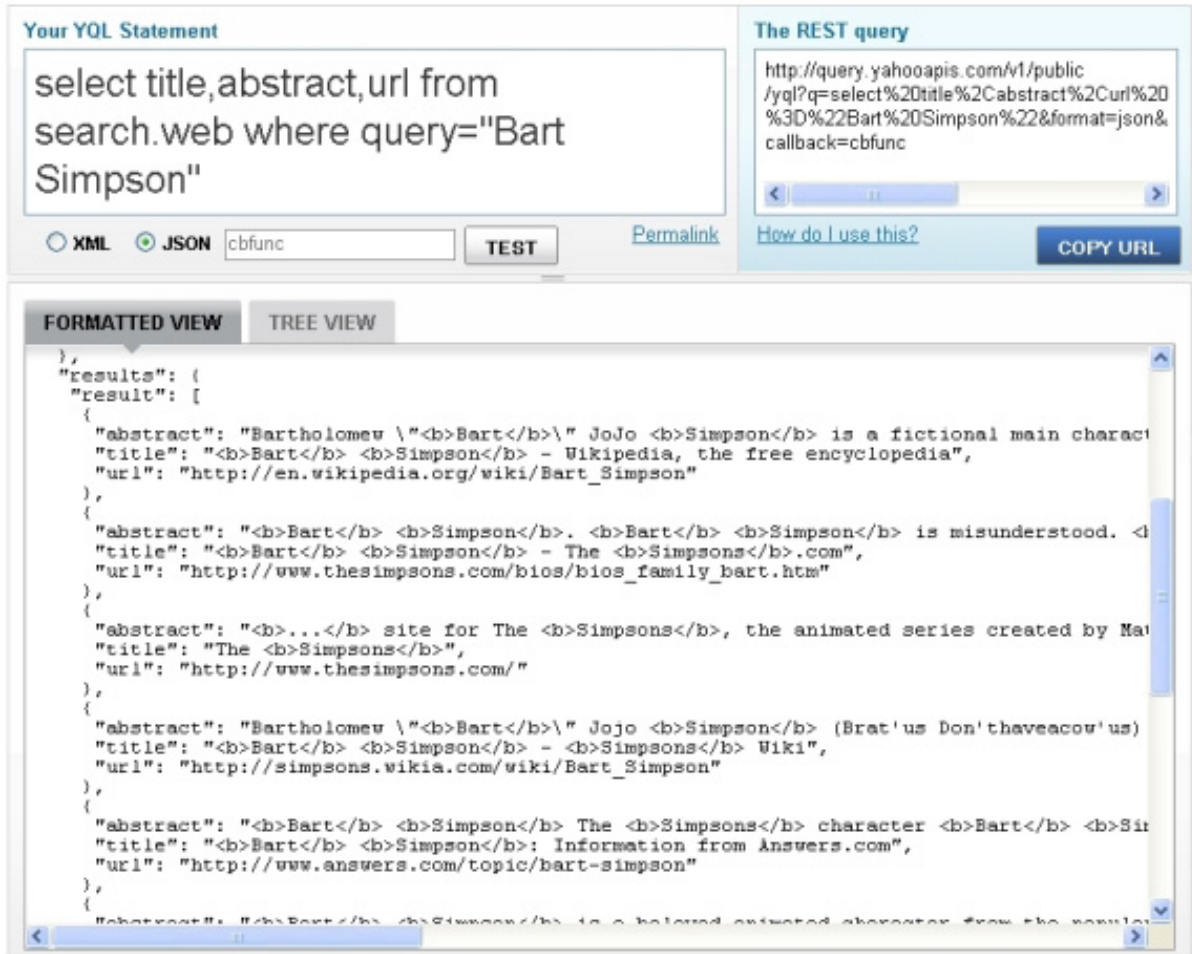
```
select title,abstract,url from search.web where query="Bart Simpson"
```

A good way to learn and improve your understanding of YQL is to use the YQL

console, which is provided by Yahoo! at the following address:
<http://developer.yahoo.com/yql/console/> . (See Figure 2 for a screen capture.)

Try to copy and paste any YQL statements you see in this article into the YQL console to examine the resulting JSONP responses (don't forget to check the radio button for getting JSON output format instead of XML).

Figure 2. YQL console



Querying YQL with jQuery

A typical YQL GET request looks like this:

```
http://query.yahooapis.com/v1/public?q=[command]&[query parameter]
```

where `command` is the YQL command and `query parameter` represents optional parameters to the service. Because you are interested in YQL's JSONP support, your requests must always include two optional parameters in order to get JSONP

responses from the YQL service. So your requests will use the following format:

```
http://query.yahooapis.com/v1/public?q=[command]&format=json&callback=?
```

where `format` is a request parameter defining the requested response format and `callback` is the name of your callback function in your Web application, which will be supplied by jQuery in this case. Note that we put a `?` as the callback function name, instead of a real function name, to let jQuery know that it should generate a function for us.

A typical call to the YQL service in jQuery looks like the code in Listing 2.

Listing 2. Typical call to YQL in jQuery

```
jQuery.getJSON(yqlUrl, function(data) {  
  // data is the JSON response  
  // process the response here  
});
```

Building the mashup

Now that you know how to query and get JSONP responses from YQL, you can start building your mashup Web page. We are now going to develop a Web page with three widgets, each of which gets its content from a different part of the Web, but through the same unified service, which is YQL.

The first widget is a stock quote widget that queries and retrieves the latest stock quotes for IBM®, Yahoo!, Google and Microsoft®. The data source for this query is Yahoo! Finance CSV files. Listing 3 shows the YQL for the first widget.

Listing 3. YQL for first widget, stock quotes

```
select symbol, price from csv  
  where url='http://download.finance.yahoo.com/d/quotes.csv?  
s=IBM,YHOO,GOOG,MSFT&f=s1d1t1c1ohgv&e=.csv' and  
  columns='symbol,price,date,time,change,col1,high,low,col2'
```

The second widget shows the headlines from *The New York Times* RSS feed, which gets its data with the YQL shown in Listing 4.

Listing 4. YQL code for second widget, RSS feed

```
select title, link from rss  
  where url="http://www.nytimes.com/services/xml/rss/nyt/HomePage.xml"
```

The third and the last widget, shown in Listing 5, is a weather forecast widget, which shows the forecast for the postcode 10504 (Armonk, NY):

Listing 5. Third and final widget, weather forecast

```
select * from weather.forecast where location=10504
```

Each widget has the skeletal code shown in Listing 6 in the Web page.

Listing 6. Skeletal code for each widget

```
<div class="widget">
  <div class="widget-head">[WIDGET HEADER]</div>
  <div class="widget-content" id="[CONTENT ID]">[WIDGET CONTENT]</div>
</div>
```

Each widget is a `div` element of class type `widget`, which has a header (`widget-head`) and content area (`widget-content`). The content area of a widget will be filled by your JavaScript code that will pull in JSONP data and append it to the content section of the widget.

You start building your mashup Web page by defining three widget placeholders: one for stock quotes, one for NYT news, and one for the weather forecast. These are shown in Listing 7.

Listing 7. Widget placeholders

```
<div class="widget">
  <div class="widget-head">Lastest stock quotes</div>
  <div class="widget-content" id="quotes"></div>
</div>
<div class="widget">
  <div class="widget-head">NYT news headlines</div>
  <div class="widget-content" id="headlines"></div>
</div>
<div class="widget">
  <div class="widget-head">Weather for Armonk NY</div>
  <div class="widget-content" id="weather"> </div>
</div>
```

Note that you leave the content areas empty because you're going to use JSONP to fill them with dynamic data. As you will shortly see, jQuery makes it very easy to generate HTML fragments on-the-fly. Each one of these areas will be filled by the skeletal jQuery code shown in Listing 8.

Listing 8. Skeletal jQuery code

```
$.getJSON(yqlUrl, function(data){
  // loop through the items
  $.each(data.query.results.[ITEM NAME], function(index, item){
    // process each item here
    // generate HTML to append to the widget's content area
  });
});
```

The code in Listing 8 issues a `GET` request to the `yqlUrl` (which you must provide)

and pulls in the JSONP response. After the JSON response arrives (which we named `data`), you get the chance to loop through the result items and process them one by one. `[ITEM NAME]` is a place holder for the array of items that may be named differently depending on your data source. Use the previously mentioned YQL console to see the resulting responses and to determine the exact response structure.

First widget

Listing 9 shows the first jQuery fragment to fill the stock quotes widget content:

Listing 9. JavaScript code of the first widget

```
var yqlUrl1= "http://query.yahooapis.com/v1/public/yql?q=
select%20symbol%2C%20price%20from%20csv%20
where%20url%3D'http%3A%2F%2Fdownload.finance.yahoo.com%2Fd%2Fquotes.csv%3F
s%3DIBM%2CYHOO%2CGOOG%2CMSFT%26f%3Ds1ld1t1c1ohgv%26e%3D.csv'%20and%20
columns%3D'symbol%2Cprice%2Cdate%2Ctime%2Cchange%2Ccol1%2Chigh%2Clow%2Ccol2'
&format=json&callback=?";
$.getJSON(yqlUrl1, function(data){
$.each(data.query.results.row, function(index, item){
$.each(item, function(i, v){
$.append(
$.('<p/>')
.append($('<span class="left"/>').text(item.symbol))
.append($('<span class="right"/>').text('$'+item.price))
);
});
});
});
```

How did we get the properly escaped URL for the first widget? When you look at the YQL console, you see a section with the title "The Rest query" on the upper right corner of the screen. After successfully testing your YQL statement, you should copy the query string from there and use it as the first parameter for the `getJSON()` function.

The stock quotes widget loops through the items and appends a `p` element that contains two `span` elements into the `div` element identified by the id `#quotes`. The resulting HTML code for the content area looks like what is shown in Listing 10.

Listing 10. HTML code of the first widget

```
<div id="quotes" class="widget-content">
<p>
<span class="left">"IBM"</span>
<span class="right"> $91.51</span>
</p>
<p>
<span class="left">"YHOO"</span>
<span class="right"> $12.22</span>
</p>
<p>
<span class="left">"GOOG"</span>
<span class="right"> $353.11</span>
</p>
```

```

</p>
<p>
  <span class="left">"MSFT"</span>
  <span class="right">$18.12</span>
</p>
</div>

```

Second widget

After finishing the first widget, we'll proceed to the second one.

Listing 11. JavaScript code of the second widget

```

var yqlUrl2= "http://query.yahooapis.com/v1/public/yql?q=
select%20title%2C%20link%20from%20rss%20
where%20url%3D%22http%3A%2F%2Fwww.nytimes.com%2Fservices%2Fxml%2Frss%2Fnyt%2F
HomePage.xml%22&format=json&callback=?";
$.getJSON(yqlUrl2, function(data){
$.each(data.query.results.item, function(index, item){
  $("<a href='" + item.link + "' target='_blank' />")
    .html(item.title)
    .appendTo($('#headlines'))
    .wrap('<p/>');
});
});

```

The code in Listing 11 first pulls in the `title` and `link` elements of the NYT RSS feed. It then appends `p` elements, containing a link to the main article, into the widget's content area. The resulting HTML code for the content area would look like the code in Listing 12.

Listing 12. HTML code of the second widget

```

<div id="headlines" class="widget-content">
  <p>
    <a target="_blank" href="http://www.nytimes.com/2009/02/19/business/19housing.html
    ?partner=rss&emc=rss">Obama Unveils $75 Billion Plan to Fight Home Foreclosures</a>
  </p>
  <p>
    <a target="_blank" href="http://www.nytimes.com/2009/02/19/business/economy/19fed.html
    ?partner=rss&emc=rss">Fed Offers Bleak Economic Outlook</a>
  </p>
  .
  .
</div>

```

Third widget

The third widget works very similar to the previous two. However, this time we append the `item.description` into the content area directly, because it already contains HTML formatted data.

Listing 13. JavaScript code of the third widget

```

var yqlUrl3= "http://query.yahooapis.com/v1/public/yql?q=
select%20*%20from%20weather.forecast%20where%20location%3D10504&
format=json&callback=?";
$.getJSON(yqlUrl3, function(data){
$.each(data.query.results.channel, function(index, item){
$.each(item.weather, function(index, item){
$.append($('
```

For the sake of completeness, Listing 14 combines everything in one place (except the CSS we used to style the widgets).

Listing 14. HTML code of the sample mashup page

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <script type="text/javascript" src="jquery-1.3.1.min.js"/>
    <style type="text/css">
      ...
    </style>
    <title>JSONP Mashup</title>
  </head>
  <body>
    <div class="widget">
      <div class="widget-head">Latest stock quotes</div>
      <div class="widget-content" id="quotes"/>
    </div>
    <div class="widget">
      <div class="widget-head">NYT news headlines</div>
      <div class="widget-content" id="headlines"/>
    </div>
    <div class="widget">
      <div class="widget-head">Weather for Armonk, NY</div>
      <div class="widget-content" id="weather"> </div>
    </div>
    <script type="text/javascript">
      var yqlUrl1="http://query.yahooapis.com/v1/public/yql?q=
      select%20symbol%2C%20price%20from%20csv%20
      where%20url%3D'http%3A%2F%2Fdownload.finance.yahoo.com%2F%2Fquotes.csv%3F
      s%3DIBM%2CYHOO%2CGOOG%2CMSFT%26f%3Dslldtltlclohgv%26e%3D.csv'%20and%20
      columns%3D'symbol%2Cprice%2Cdate%2Ctime%2Cchange%2Ccoll%2Chigh%2Clow%2Ccol2'
      &format=json&callback=?";
      $.getJSON(yqlUrl1, function(data){
        $.each(data.query.results.row, function(index, item){
          $('#quotes')
            .append(
              $('
```

```

$.each(data.query.results.item, function(index, item){
  $("<a href='" + item.link + "' target='_blank'>")
    .html(item.title)
    .appendTo($('#headlines'))
    .wrap('<p/>');
});
});

var yqlUrl3= "http://query.yahooapis.com/v1/public/yql?q=
select%20*%20from%20weather.forecast%20where%20location%3D10504&
format=json&callback=?";
$.getJSON(yqlUrl3, function(data){
  $.each(data.query.results.channel, function(index, item){
    $('#weather')
      .append('<p/>').html(item.description);
  });
});
</script>
</body>
</html>

```

Figure 3 shows a screenshot of the resulting Web page when this article was written.

Figure 3. Sample mashup page



Conclusion

YQL is a powerful service that makes client-side mashups possible without using server-side proxies. Coupled with its JSONP support and jQuery, YQL lets you access structured data across the Web through a single unified interface. In this article, you learned how to use jQuery and YQL to build up a mashup Web page with few lines of code. You can use this as a beginning and improve the code to make it more sophisticated. Here are some tips:

- Put an image link into the widget header that refreshes the content area when clicked.
- Provide an input field for the user to enter stock symbols and fetch the quotes for the entered symbols only.
- Show not only RSS items' title and link, but also their descriptions.

Resources

- ["Cross Domain Communications with JSONP: Part 1"](#) (developerWorks, February 2009): Read the first part of this article series, which introduces JSONP.
- [Yahoo! Query Language](#): Get to know YQL from its inventors.
- [YQL Console](#): Try the YQL console to learn about and get practice with the YQL.
- ["Working with jQuery, Part 1: Bringing desktop applications to the browser"](#) (developerWorks, September 2008): Get to know jQuery in this three-part article series.

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

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.