

We've all seen [Google finance](#), and the great job that it does at mixing HTML content with Flash content, as seen on [company stock quotes](#). Google has done a great job at using Flash where it makes the most sense, in the graphs for company quotes, with HTML where it makes sense (linking to news items). Today, I'll show you how to build your own Google-finance type site, using a combination of Ajax, JSON, PHP and Flex / Flash. And, best of all, **we'll do it all for free.**

In this example, we'll grab data from a Feedburner feed, and then show the hits to that feed in a graph, over time. We'll bring in the RSS information from our blog feed, and when we click on the items in the graph, they'll highlight the RSS feed items for that day.

To complete this tutorial, you'll need the following software / accounts:

<i>Software / Account</i>	<i>Why do we need it?</i>	<i>Cost?</i>	<i>Where do I get it?</i>
Feedburner with API access turned on	Data to populate an Ajax page and Flash graph.	\$0	www.feedburner.com
PHP	Acts as a proxy for our Ajax application. Connects to Feedburner and prints out XML.	\$0 / Open Source	www.php.net
Spry Framework for Ajax	Create the Ajax page	\$0 / Open Source	labs.adobe.com
Flex and Flex charting	Create the .swf file that will show the graph and call back to JavaScript	Flex SDK: Free Flex Charting: Free trial.	www.adobe.com/go/flex
Flex / Ajax bridge	Call ActionScript (Flash) from JavaScript	\$0 / Open Source	labs.adobe.com
ActionScript 3 Library for JSON	Parse the JSON that we're passing to Flex	\$0 / Open Source	labs.adobe.com
JSON and JSMin Library	Remove new lines and carriage returns in JSON	\$0 / Open Source	

1. Note: You can perform this same tutorial without Flex Charting. You would need to create your own Flex component, and you could do that with the SDK. So, technically this tutorial could be accomplished for \$0. In my case, to reduce development time, I'm using the Flex Charting components, which cost \$249.

The first part to this tutorial is to install all the above software. I won't go through that here, if you have problems please comment in my blog.

Believe it or not, there actually are pieces of Adobe software that I don't use, feel free to use Photoshop to design the appearance of the webpage, Lightroom to hold photos of people you will put on the HTML page, Premiere to do video editing, ColdFusion can replace the PHP part fairly easily etc... I leave it up to you to extend this tutorial to make use of every piece of Adobe software.

Once we've got everything installed, we'll start by hooking up our PHP backend to Spry. The reason that we need the PHP backend is because Ajax applications cannot load data from outside sources: we

need to load data from the same source as the Ajax application. So, we've created a small PHP file that will go to Feedburner, get the stats and then print the output. That PHP file is very small and very simple:

```
<?php
$fpURL =
'http://api.feedburner.com/awareness/1.0/GetFeedData?uri=adobe/mpotter&dates=2006-
07-01,2006-07-17';
$handle = fopen($fpURL, "r");
while (!feof($handle)) {
    $strOutData .= fread($handle, 8192);
}
fclose($handle);
header('Content-type: text/xml');
echo $strOutData;
?>
```

Any server side language could do something similar.

And here's the JavaScript code for Spry to call and load that PHP file:

```
var dsFeedburner = new Spry.Data.XMLDataSet("getdata.php", "/rsp/feed/entry");
```

When the HTML page loads, it will call getdata.php, and populate the Spry dsFeedburner data source with data from that file. Here's a sample of the XML content that gets output from that PHP file.

```
<?xml version="1.0" encoding="UTF-8"?>
<rsp stat="ok">
  <!--This information is part of the FeedBurner Awareness API. If you want to hide
this information, you may do so via your FeedBurner Account.-->
  <feed id="412263" uri="adobe/mpotter">
    <entry date="2006-07-01" circulation="0" hits="0"/>
    <entry date="2006-07-02" circulation="0" hits="0"/>
  ...
```

The text `"/rsp/feed/entry"` is simply an xpath expression to get to each `<entry...>` item in the XML file.

So, now we've got the data. To populate a table with data, we do the following:

```
<table border="1" spry:region="dsFeedburner">
  <tr>
    <th onClick="dsFeedburner.sort ('{@date}');">Date</th>
    <th onClick="dsFeedburner.sort ('{@hits}');">Hits</th>
  </tr>
  <tr onclick="alert ({@date});" spry:repeat="dsFeedburner"
spry:select="SelectedFeedburnerItem" spry:hover="HoverFeedburnerItem">
    <td>{@date}</td>
    <td>{@hits}</td>
  </tr>
</table>
```

We repeat each table row (`<tr>`) with the `spry:repeat="dsFeedburner"` attribute. We set the selected item's CSS class to `"SelectedFeedburnerItem"`, and the hover state to `"HoverFeedburnerItem"`. You can edit the style for those items with simple CSS, in the HTML file (or an external CSS file, whichever you prefer.)

OK, so if we run that, then we should see a repeatable table showing dates and hits to our items. Pretty good, now let's hook that up to a graph component that we made in Flex.

Its important to note here that Flex can be used to create a number of components. Its possible, with a little work, that you could create a component that exactly matches the graphing component that Google uses, with a date slider / selector at the top, above the graph. I won't do that here, for the sake of simplicity, but it is possible. Or, you could create an Ajax media browser that plays videos in Flash. The possibilities are limited only by your imagination.

So, let's go ahead and create our Flex graph... Here's the MXML code:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
width="400" height="400" backgroundGradientColors="[#ffffff, #ffffff]">
  <fab:FABridge xmlns:fab="bridge.*" />
  <mx:Script>
    <![CDATA[
      import mx.charts.events.ChartItemEvent;
      import mx.collections.ArrayCollection;
      import com.adobe.serialization.json.JSON;

      public function bindJSONToChart( JSONString:String ):void {
        var arr:Array = (JSON.decode(JSONString) as Array);

        linechart.dataProvider = new ArrayCollection(arr);
      }

      public function chartClicked(clickEvent:ChartItemEvent):void
      {
        if( ExternalInterface.available )
          ExternalInterface.call("chartClicked",
clickEvent.hitData.item);
      }
    ]]>
  </mx:Script>
  <mx:LineChart id="linechart"
paddingLeft="5" paddingRight="5"
showDataTips="true" left="10" right="10" top="10" bottom="10"
itemClick="callEI(event)">

    <mx:horizontalAxis>
      <mx:CategoryAxis categoryField="@date" displayName="Date"/>
    </mx:horizontalAxis>

    <mx:series>
      <mx:LineSeries yField="@hits" displayName="Hits"/>
    </mx:series>
  </mx:LineChart>
</mx:Application>
```

Everything should be fairly self-explanatory. The only lines that may be unfamiliar are:

```
<fab:FABridge xmlns:fab="bridge.*" />
```

which is needed for the Flex / Ajax bridge to function properly and:

```
public function callEI(func:String, clickEvent:ChartItemEvent):void
{
  if( ExternalInterface.available )
    ExternalInterface.call("chartClicked", clickEvent.hitData.item);
}
```

which is needed to call back to the page that contains the Flash file, in our case the Ajaxed HTML

page.

OK, so we've got our Flex application built, we've got our Ajax file, now we need to pass data from the Ajax page to the Flex application. Here's how you do that in JavaScript:

```
var obs = new Object;
obs.onPostLoad = function(notifier, data)
{
    Spry.Debug.trace("obs.onPostLoad called!");
};

var notifierData;

obs.onDataChanged = function(notifier, data)
{
    notifierData = notifier.data;

    try {
        with( FABridge.example.root() ) {
            bindJSONToChart( jsmin( toJsonString( notifier.data ) ) );
        }
    }
    catch( e )
    {
        var initCallback = function( )
        {
            with( FABridge.example.root() ) {
                bindJSONToChart( jsmin( toJsonString( notifierData ) ) );
            }
        }
        FABridge.addInitializationCallback("example",initCallback);
        //alert( "Error in onDataChanged"+e );
    }
    Spry.Debug.trace(toJsonString( notifier.data ));
};

dsFeedburner.addObserver( obs );
```

There's lots going on here. First, we create an observer on the data's "onDataChanged" method, so every time the data is changed, we send new data to the Flex application. Although we only use this once, when the page is loaded, you could use this if you had say a drop down list of a number of your feeds, and wanted the data source to change when the drop down changed. When you changed your Spry data source, the Flex graph would refresh because of this observer.

Then, we hack around a bit. First of all, we need to convert the data to JSON format:

```
toJsonString( notifier.data )
```

We also need to minimize it, because Flex's JSON Library doesn't like new lines or carriage returns in that data (this took me a long time to figure out!). So, we run:

```
jsmin( toJsonString( notifier.data ) )
```

Finally, we pass the result of that function to the method in our MXML file, `bindJSONToChart()`:

```
bindJSONToChart( jsmin( toJsonString( notifier.data ) ) );
```

I'm lazy and I've made that all one line in the code.

Now, it will do that fine if the .swf file has loaded. However, sometimes the data gets returned before the .swf file is actually loaded. I found this out when running the application locally. So, I've wrapped

all this in a try / catch statement, and added an initialization function on the swf file, so that if its not loaded, when it loads it will run this function and populate the graph with data from the Ajax call.

There you go. When you load the HTML page, Spry will load that PHP file using an Ajax call, the PHP file will connect to Feedburner, get the XML data, print it out. Spry will read that in, bind it to the HTML elements on the page, then call the Flex application, passing it the data in JSON format. The Flex application will read in the data and display it in a Flex chart.

Here are a few tips and tricks that I've found when building this out:

1. I found it easiest to modify the HTML that gets output by Flex Builder, rather than to reference the built swf file. If you do that, be sure to modify index.template.html in the html-template folder of your Flex project, rather than the .html files in the bin/ directory of your Flex project. The HTML files in bin/ get overwritten when you save and re-build your Flex application, and if you modify those, rather than index.template.html, you'll lose your changes.
2. I started building out the Flex graph using JavaScript and the Flex / Ajax bridge. I don't recommend that. Build out your entire Flex component in Flex Builder, then simply write functions to pass data to it from HTML. I think that's easier than trying to build Flex components using JavaScript.
3. Similarly, write functions that closely couple your Flex application to your HTML page. For instance, you'll notice that in my MXML file, I call the chartClicked JS function using:

```
if( ExternalInterface.available )  
    ExternalInterface.call("chartClicked", clickEvent.hitData.item);
```

I could have attached an observer to the lineChart instead, using the Flex / Ajax bridge, but I find it easier to get the data items and debug the application in Flex Builder, rather than trying to do that in JavaScript on the HTML page.