

**Seattle Activities Web Service**

Creator: Patrice Briggs

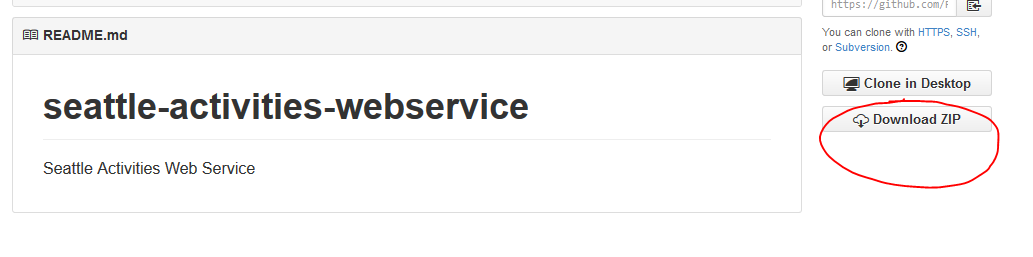
Date: 2/24/2015

Below is a link to my Seattle Activities web service. All code required for it to run on a PHP enabled web server is included.

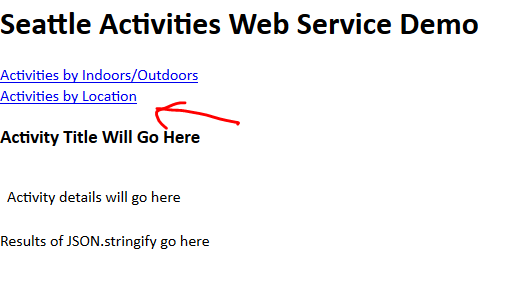
**Demo:** A demo page implementing the web service is here:<http://web-students.net/site7/Sandbox/Web%20Services/Seattle%20Activities/SeattleActivitiesindex.htm>

**Download the Code:** Code is to be found in the Github repo: <https://github.com/PatriceBriggs/seattle-activities-webservice>

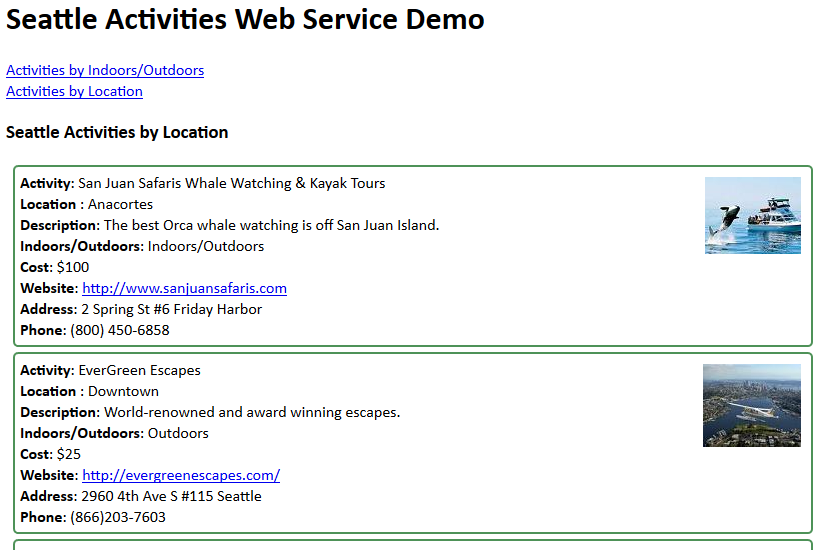
To download the file, find the zip icon in the bottom right hand corner of the screen:

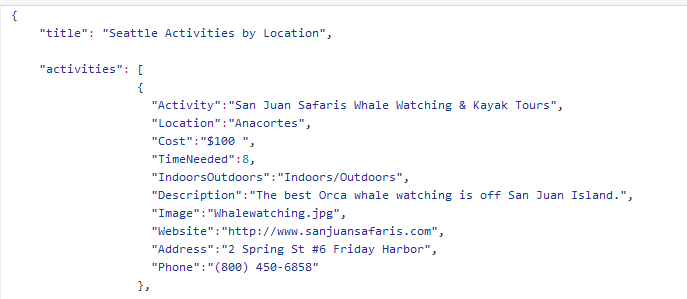


**Upload and test:** Upload the entire folder to a PHP enabled web server. There is a test page, named **SeattleActivitiesindex.htm** in the root of the folder. It looks like the following:

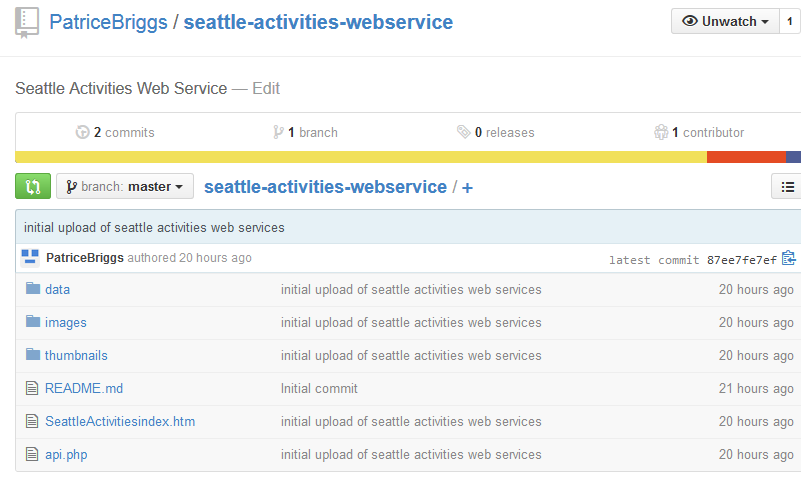


Clicking the links above will trigger the page to use AJAX to retrieve and parse one of two JSON files. Here is some sample output:

In the above, an <h3> which prior to the AJAX call contained the text “Activity Title Will Go Here” has been replaced with data from the JSON file. Here is a snapshot of the JSON file:



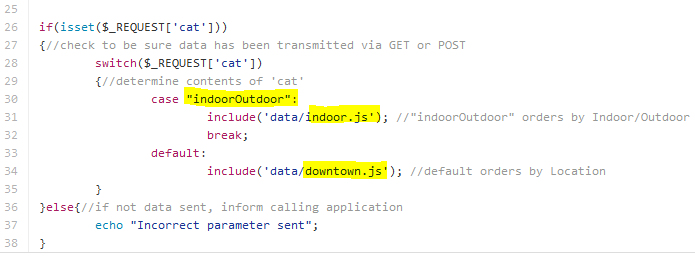
Above you can see the “**title**” and an array of “**activities**” and several properties for each activity. Not all of the properties above are implemented in the demo, for instance, the value of **“Image**”, for example **Chihuly.jpg**, applies both to a large image in a folder named **images** and a smaller image in a folder named **thumbnails**. Here is a snapshot of the directory structure of the folder:



Above you can see the **images** and **thumbnails** folders, both of which contain a set of **jpgs** of each activity, with identical file names. The **data** folder contains two JSON files, one named **downtown.js**, which depicts 34 Seattle area activities ordered by the location of the activity and **indoor.js**, which is the same data sorted in order whether it is an indoor, an outdoor or an indoor/outdoor activity. The third file, **SeattleActivities.xls**, is the spreadsheet from which most of the original data for the films was derived.

The other two files visible, are the main API file that is called via AJAX, named **api.php** and the demo file named **SeattleActivitiesindex.htm**.

**API File:** The file that receives the AJAX call and manages the JSON files is named **api.php**:



In the above file, data is passed via GET or POST to the web page hosted on a web server. The parameter “**cat**” (short for category) contains a small string that will indicate which category to show. If the “**cat**” contains the string “**IndoorOutdoor**” as it does below, the file **indoor.js** will be loaded:

**api.php?cat=indoorOutdoor**

If any other string is sent as the contents of the parameter “cat”, the file named **downtown.js** will be loaded:

**api.php?cat=Location**

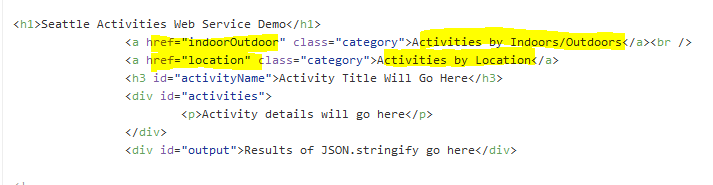
If no data is sent, for example:

**api.php?**

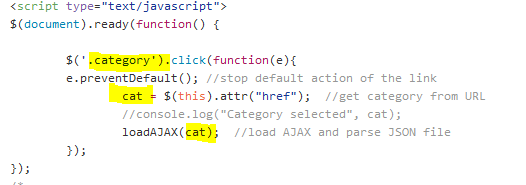
Then the following message will be returned:

**Incorrect parameter sent**

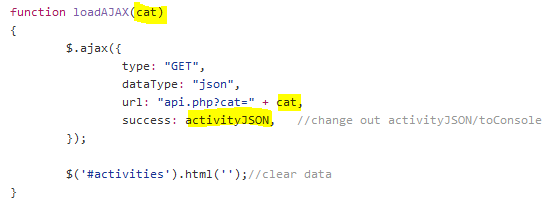
**Demo file:** The file that does all of the AJAX work is named **SeattleActivitiesindex.htm**. This file has links that are loaded to use AJAX to call **api.php**, pass a parameter to it, and when JSON data is returned, parse it for output on the page. Inside the page, the links are loaded with data in the **href** attribute that is used to indicate which JSON file to load:



These are wired with click events:

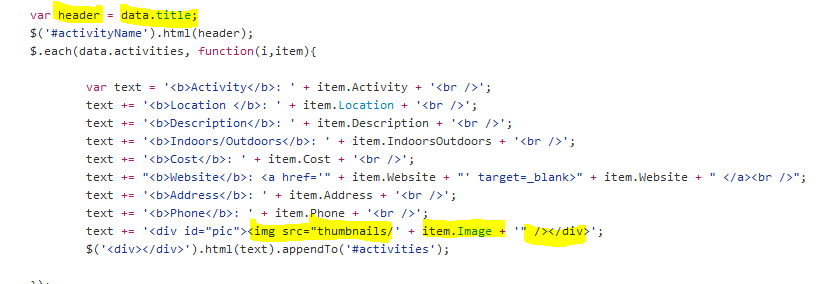


The class of **category** is used to determine which links get the click treatment. The attribute data is pulled from the **href** and loaded into a function named **loadAJAX()** which calls the API page.



The data from the **href** is inside the variable named **cat** which is loaded via GET to the **api.php** page. Upon successful return, the function named **activityJSON()** will parse the JSON data:

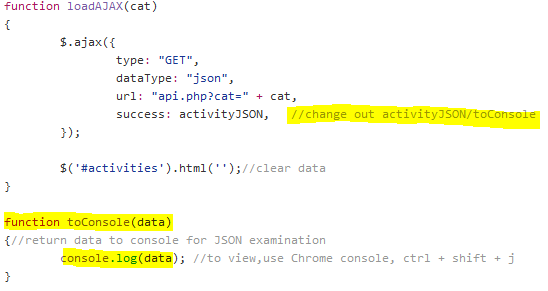




The attribute **title** is loaded into an **<h3>** with an id of **activityName** above the actitivites. In the JSON object an array named “**activities”** is looped via jQuery’s **each()** method and individual data items are created and inserted into a new div and then appended to another **<div>** with an id of **activities**.

Note the JSON **Image** attribute is loaded into the **src** attribute of an **img** element and positioned absolutely right in the **<div>**. The thumbnail is used here instead of loading all the large images that would drag down the speed of the page.

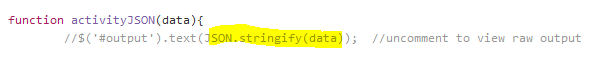
**Troubleshooting:** Our first line of defense in troubleshooting comes from the **Chrome** (not Firefox) console. We have a special function placed in our code that will pass the raw returned JSON output as an object into the console:



Above we can replace the **activityJSON()** function with a new function named **toConsole()**. This function passes the JSON data returned into Chrome’s console, allowing us to examine the data as an actual JS object. Use the shortcut keys **ctrl + shift + j** (**command + shift + j** on a mac) in order to view the data via the console:

In the console, you can see the two properties in the object, the **activityName** and the **activities** array. By clicking the activitiesarray, you can examine the data further. In this way we can examine the API data to be returned without a great deal of advance knowledge of the structure.

Another way to see the data is to dump it to the screen. There is a line commented out in the demo:



When un-commented, it uses the powerful [JSON.stringify()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify) to attempt to show the raw JSON data returned. You can also use this to verify the data you’re looking for has the proper attributes.

**Ways to use the API:** When integrating the API in your pages, there are several things that you could do with the data. For example, there are larger versions of the image that could be popped into a jQuery based modal window, such as is used by [Lightbox 2](http://lokeshdhakar.com/projects/lightbox2/).