JavaScript + PHP / AJAX

Costantino Pistagna
<pistagna@dmi.unict.it>

What's is Ajax?

- **AJAX** is not a new programming language
  - It's a new technique for better and faster web applications.
  - **JavaScript** can communicate directly with the server.
    - with the **XMLHttpRequest** object.
- **JavaScript** can trade data with webServer
  - without reloading the page.

Ajax Technologies

- AJAX uses asynchronous data transfer
  - HTTP requests
  - between the browser and the web server
- Request small bits of information from the server
  - instead of whole pages
  - speed++
  - interaction
- AJAX makes Internet applications
  - smaller, faster and more user-friendly.

AJAX Standards

- AJAX is based on the following web standards:
  - JavaScript
  - XML / JSON
  - HTML
  - CSS

The XMLHttpRequest object

- To get or send information from/to a server with **traditional JavaScript**
  - Make an HTML form
  - Click the "Submit" button to send/get the information
  - Wait for the server to respond
- Then a new page will load with the results
  - The server returns a new page each time the user submits input
    - Traditional web applications can run slowly
    - Tend to be less user-friendly

The XMLHttpRequest object

- With AJAX, your JavaScript communicates directly with the server,
  - Through the JavaScript XMLHttpRequest object.
- XMLHttpRequest makes a request and get a response from server
  - without reloading the page.
- The user will stay on the same page
  - he will not notice that scripts request pages
  - data to server in background.
- Introduces some things that were otherwise impossible
  - HEAD requests

How does it work?

- Instead of a user request being made to server via
  - HTTP POST or GET request
  - such as would be made by submitting a form
- An Ajax script makes a request to the server
  - by using the Javascript XMLHTTPRequest object
- This object may be unfamiliar to many
  - it behaves like a fairly ordinary javascript object

How does it work?

- When using a **javascript** image object...
  - We may **dynamically** change the URL of the image source
    - Without using a page refresh.
  - **XMLHTTPRequest** retrieves information from the server in a similarly **invisible** manner.

Why XML HTTP Request object?

- Whilst the object is called the **XML HTTP Request** object
  - it is not limited to being used with **XML**
  - it can request or send any type of document
- Dealing with binary streams can be problematical in javascript.
  - we will see the **JSON** alternative

Ajax Events

- Ajax uses a programming model with
  - display and events
- These events are user actions
  - they call functions associated to elements of the web page
- Interactivity is achieved with forms and buttons.
- DOM allows to link elements of the page with actions and also to extract data from XML files provided by the server.

XMLHttpRequest Methods

- To get data on the server, XMLHttpRequest provides two methods:
  - **open**: create a connection.
  - **send**: send a request to the server.
- Data by the server will be found in the attributes of the XMLHttpRequest object:
  - **responseXml** for an XML file or
  - **responseText** for a plain text.
- Take note that a new XMLHttpRequest object has to be created for each new file to load.
- We have to wait for the data to be available to process it
  - The state of availability of data is given by the **readyState** attribute of XMLHttpRequest.

How is it coded?

- We need to know how to create an XMLHTTPRequest object.
- The process differs slightly
  - if you are using Internet Explorer (5+) with ActiveX enabled
  - or a standards-compliant browser such as Mozilla Firefox.
- With IE, the request looks like:

```
http = new ActiveXObject("Microsoft.XMLHTTP");
```

- In a standards browser we can instantiate the object directly:

```
http = new XMLHttpRequest();
```

Our Implementation

```
function getHTTPObject() {
  if (typeof XMLHttpRequest != 'undefined') {
    return new XMLHttpRequest();
  }
  try {
    return new ActiveXObject("Msxml2.XMLHTTP");
  } catch (e) {
    try {
      return new ActiveXObject("Microsoft.XMLHTTP");
    } catch (e) {}
  }
  return false;
}
```

Our Implementation

- We need to write an event handler
  - which will be called via some event on our user's page
  - Will handle sending our request for data to our server
- The event handler will use methods of XMLHTTPRequest object:
  - **Make** the request of the server
  - **Check** when the server has completed the request
  - **Deal** with the information returned by the server

Our Implementation

- For example,
  - We can make our request of the server by using a GET method to an appropriate server-side script.
  - Assuming that we have created our XMLHTTPRequest object and called it **http**:

```
var http = getHTTPObject();
http.open("GET", "http://example.org/test.txt", true);
http.onreadystatechange = function() {
   if (http.readyState == 4) {
      doSomethingWith(http.responseText);
   }
}
http.send(null);
```

Our Implementation

- The function listens to the **onreadystatechange** property
  - each time this parameter changes
    - calls a further function
- We said little about the server-side script which is called
  - Essentially this can be any server routine which will generate the required output when called with the relevant URL and appended parameters
  - as in any other HTTP GET request.

Handling data: readyState

- We need to write a function useHttpResponse
  - will establish when the server has completed our request
  - and do something useful with the data it has returned:

```
function useHttpResponse() {
  if (http.readyState == 4) {
    var textout = http.responseText;
    document.write.textout;
  }
}
```

Handling data: readyState

- Our function checks for a **readyState** value of 4
- there are various numbered states describing the progress of such a request
- tipically, we are interested in the value of 4
  - which indicates that the request is complete and we can use the returned data.
  - ```
    0: not initialized.
    1: connection established.
    2: request received.
    3: answer in process.
    4: finished.
    ```
    ```

responseText

```
var textout = http.responseText;
```

- We have received our information as simple text
  - via the responseText property of our XMLHTTPRequest object
- Information can be returned as XML
- or as properties of a predefined javascript object

Example: Get a text

```
<html><head><script>
function submitForm(){
   var http;
   try {  http = new ActiveXObject('Msxml2.XMLHTTP');   }
   catch (e) {
      try {   http = new ActiveXObject('Microsoft.XMLHTTP');    }
      catch (e2) {
         try {  http = new XMLHttpRequest();     }
         catch (e3) {  http = false;   }
      }
   }
   http.onreadystatechange  = function() {
      if(http.readyState  == 4) {
         if(http.status  == 200)
            document.ajax.dyn="Received:"  + http.responseText;
         else
            document.ajax.dyn="Error code " + http.status;
      }
   };
    http.open(GET, "data.txt",  true);
    http.send(null);
}
</script></head>
<body>
    <FORM method="POST" name="ajax" action="">
         <INPUT type="BUTTON" value="Submit"  ONCLICK="submitForm()">
         <INPUT type="text" name="dyn"  value="">
    </FORM>
</body></html>
```

Get from XML

- To get data from an XML file, we have just to replace this line:

```
document.ajax.dyn="Received:"  + http.responseText;
```

- by this code:

```
// Assign the XML file to a var
var doc = http.responseXML;

// Read the first element
var element = doc.getElementsByTagName('root').item(0);

// Assign the content to form
document.ajax.dyn.value= element.firstChild.data;
```

Write to body

- In this demo, the text read is put into the body of the page
  - not into a textfield
- The code below replaces the textfield form object

```
<div id="zone">
     ... some text to replace ...
 </div>
```

- the second part replaces the assignment into the JavaScript function.

```
document.getElementById("zone").innerHTML = "Received:" + http.responseText;
```

Post a text

- In this demo, a text is sent to the server and is written into a file.
- The call to the "open" method changes
  - the argument is POST
  - the url is the name of a file or script that receives the data sent
  - the "send" method has now a value as argument that is a string of parameters.

```
http.open("POST", "ajax-post-text.php", true);
http.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
http.send(data);
```

Post a text

- The parameter of the send method is in the format of the HTML POST method.
- When several values are sent, they are separated by the ampersand symbol:

```
var data = "file=" + url + "&content=" + content;
```

A sample and full application - Step #1

- Creating the form
  - We first create a simple webpage that has the HTML for our Web form.
  - There is nothing out of the ordinary here - just basic HTML defining the city, state, and ZIP code.

```
<!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>
<title>ZIP Code to City and State using XmlHttpRequest</title>
</head>
<body>
<form action="post">
  <p>
  ZIP code:
  <input type="text" size="5" name="zip" id="zip"  />
  </p>
  City:
  <input type="text" name="city" id="city" />
  State:
  <input type="text" size="2" name="state" id="state" />
</form>
</body></html>
```

A sample and full application - Step #2

- Add an **onblur** event handler function named updateCityState()
  - This event handler is called whenever the ZIP code field loses focus.
  - onblur="updateCityState();"
  - The updateCityState() function will be in charge of asking the server what the city and state is for a given Zip code.
  - For now, this function does nothing.

```
...
<script language="javascript"  type="text/javascript">
function updateCityState() { /* does nothing...yet! */}
</script>
</head><body>
<form action="post">
<p>ZIP code:
<input type="text" size="5" name="zip" id="zip" onblur="updateCityState();" />
...
```

A sample and full application - Step #3

- Of course we need to create an XMLHttpRequest object.

```
function getHTTPObject() {
  var xmlhttp;
  if (!xmlhttp && typeof XMLHttpRequest != 'undefined') {
    try {
      xmlhttp = new XMLHttpRequest();
    } catch (e) {
      xmlhttp = false;
    }
  }
  return xmlhttp;
}
var http = getHTTPObject(); // We create the HTTP Object
```

A sample and full application - Step #4

- Now add the code that makes the round trip to the server.
- We first specify the URL for the server-side script:
- getCityState.php?param=
- This URL will then have the ZIP Code appended to it so that the ZIP code is passed as an HTTP GET parameter named param
- This means that if a user types in the ZIP code of 17534, the resulting URL would be getCityState.php?param=17534.
- Before we send the HTTP request, we specify the onreadystatechange property
  - The name of our new function handleHttpResponse().
  - This means that every time the HTTP ready state has changed, our function handleHttpResponse() is called.

A sample and full application - Step #4

- Our new function handleHttpResponse() sits there waiting to be called and when it does get called, it check to see if the readState is equal to 4.
  - If it is equal to 4 this means that the request is complete.
- Since the request is complete, it is now fair game to grab the response text (responseText)
  - unpack it,
  - set the city and state form fields to the returned values

```
var url = "getCityState.php?param="; // The server-side script
function handleHttpResponse() {
  if (http.readyState == 4) {
    // Split the comma delimited response into an array
    results = http.responseText.split(",");
    document.getElementById('city').value = results[0];
    document.getElementById('state').value = results[1];
  }
}

function updateCityState() {
  var zipValue = document.getElementById("zip").value;
  http.open("GET", url + escape(zipValue), true);
  http.onreadystatechange = handleHttpResponse;
  http.send(null);
 }
```

A sample and full application - Step #5

- Lets now create a PHP file named getCityState.php
  - All this file does is return “Catania, Italy” as the city and state.
  - This means that anytime the focus leaves the ZIP code field
    - The city and state will be automatically set to Catania, Itay

```
...
<?php echo "Catania, Italy"; ?>
...
```

- **@home:** get results from a text file with the following format

```
...
CITY:STATE
...
```

AJAX Suggest

- HTML Part:

```
<form>
First Name: <input type="text" id="txt1"
                                onkeyup="showHint(this.value)"/>
</form>

<p>Suggestions: <span id="txtHint"></span>
</p>
```

AJAX Suggest

- JavaScript Part:

```
function showHint(str) {
if (str.length==0) {
  document.getElementById("txtHint").innerHTML="";
  return;
}
http=GetXmlHttpObject();
if (http==null) {
  alert ("Your browser does not support XMLHttpRequest!");
  return;
}
var url="gethint.php";
url=url+"?q="+str;
url=url+"&sid="+Math.random();
http.onreadystatechange=stateChanged;
http.open("GET",url,true);
http.send(null);
}
```

AJAX Suggest

- JavaScript Part2:

```
function stateChanged() {
   if (http.readyState==4) {
      document.getElementById("txtHint").innerHTML=http.responseText;
  }
}
```

AJAX Suggest

- PHP Part:

```
<?php
// Fill up array with names
$a[]="Anna";$a[]="Barbara";$a[]="Cinzia";$a[]="Diana";$a[]="Elisa";
$//get the q parameter from URL
$q=$_GET["q"];
//lookup all hints from array if length of q>0
if (strlen($q) > 0) {
  $hint="";
  for($i=0; $i<count($a); $i++) {
    if (strtolower($q)==strtolower(substr($a[$i],0,strlen($q)))){
      if ($hint==""){
        $hint=$a[$i];
      } else {
        $hint=$hint." , ".$a[$i];
      }
    }
  }
}
// Set output to "no suggestion" if no hint were found
if ($hint == "") {
  $response="no suggestion";
} else {
  $response=$hint;
}
//output the response
echo $response;
?>
```

JSON: What does it stand for?

- JavaScript Object Notation
  - JSON is a syntax for passing around objects that
    - contain name/value pairs
    - arrays
    - other objects

```
{"skillz": {
    "web":[
        {"name": "html",
         "years": "5"
        },
        {"name": "css",
         "years": "3"
        }]
    "database":[
        {"name": "sql",
         "years": "7"
        }]
}}
```

Squiggles, Squares, Colons and Commas

- Squiggly brackets act as 'containers'
- Square brackets holds arrays
- Names and values are separated by a colon.
- Array elements are separated by commas
- Think 'XML with Anorexia'

JSON is like XML because:

- They are both 'self-describing' meaning that values are named, and thus 'human readable'
- Both are hierarchical. (i.e. You can have values within values.)
- Both can be parsed and used by lots of programming languages
- Both can be passed around using AJAX (i.e. httpWebRequest)

JSON is UNlike XML because:

- XML uses angle brackets, with a tag name at the start and end of an element: JSON uses squiggly brackets with the name only at the beginning of the element.
- JSON is less verbose so it's definitely quicker for humans to write, and probably quicker for us to read.
- JSON can be parsed trivially using the eval() procedure in JavaScript
- JSON includes arrays {where each element doesn't have a name of its own}
- In XML you can use any name you want for an element, in JSON you can't use reserved words from javascript

The Object: An Introduction

- Behold, an Object

```
var myFirstObject = {};
```

- The old way to create a new object was to use the "new" keyword.

```
var myJSON = new Object();
```

- This method has been deprecated now in favor of simply defining an empty object with squigly braces.

```
var myJSON = {};
```

Objects as Data

- A Javascript Object is a very flexible and robust data format expressed as
  - “name/value pairs”
- That is, an object holds a name which is an object's property
  - Think of it as a plain old variable name that's attached to the object name.
  - And the object holds the value of that name

```
var myFirstJSON = { "firstName" : "Mario",
                    "lastName"  : "Rossi",
                    "age"       : 23 };

document.writeln(myFirstJSON.firstName);  // Outputs Mario
document.writeln(myFirstJSON.lastName);   // Outputs Rossi
document.writeln(myFirstJSON.age);        // Outputs 23
```

Objects as Data

```
var myFirstJSON = { "firstName" : "Mario",
                    "lastName"  : "Rossi",
                    "age"       : 23 };
```

- This data format is called **JSON** for **J**ava**S**cript **O**bject **N**otation.
- What makes it particularly powerful is that since the value can be any data type
  - you can store other arrays and other objects
    - nesting them as deeply as you need

Objects as Data

```
var employees = { "accounting" : [   // accounting is an array in employees.
                                    { "firstName" : "Mario",  // First element
                                      "lastName"  : "Rossi",
                                      "age"       : 23 },

                                    { "firstName" : "Luca",  // Second Element
                                      "lastName"  : "Verdi",
                                      "age"       : 32 }
                                  ], // End "accounting" array.
                  "sales"       : [ // Sales is another array in employees.
                                    { "firstName" : "Mario", // First Element
                                      "lastName"  : "Biondi",
                                      "age"       : 27 },

                                    { "firstName" : "Marco",   // Second Element
                                      "lastName"  : "Arancio",
                                      "age"       : 41 }
                                  ] // End "sales" Array.
                } // End Employees
```

- **employees** is an object.
  - That object has two properties or name/value pairs.
- **accounting** is an array which holds **two JSON** objects showing the names and age of 2 employees.
- **sales** is also an array which holds **two JSON** objects showing the name and ago of the two employees who work in sales.
- All of this data **exists within** the employees object.

Accessing Data In JSON

- The most common way to access JSON data is
  - through dot notation
  - This is simply the object name followed by a period and then followed by the name/property you would like to access.

```
var myObject = { 'color' : 'blue' };

document.writeln(myObject.color); // outputs blue.
```

  - If your object contains an object then just add another period and name

```
var myObject = { 'color' : 'blue',
                 'animal' : {'dog' : 'friendly' }
               };

document.writeln(myObject.animal.dog); // outputs friendly
```

Accessing Data In JSON

- Using the "employee" example above
  - if we wanted to access the first person who worked in sales

```
document.writeln(employees.sales[0].firstName + ' ' +
employees.sales[0].lastName);
```

  - We can also access the second person who works in "accounting"

```
document.writeln(employees.accounting[1].firstName + ' ' +
employees.accounting[1].lastName);
```

Recap

- to recap...
  - The “employee” example is an object which holds two arrays
    - each of which holds two additional objects.
  - The only limits to the structure are the amount of storage and memory available to it.
  - Because JSON can store objects within objects within objects and arrays within arrays that can also store objects
    - There is no virtual limit to what a JSON object can store.
  - Given enough memory and storage requirement, a simple JSON data structure can store, and properly index, all the information ever generated by humanity.

Simulating An Associative Array

- You can also access JSON data as if it were an Associative Array.

```
var myFirstJSON = { "firstName" : "Mario",
                    "lastName"  : "Rossi",
                    "age"       : 23 };

document.writeln(myFirstJSON["firstName"]);  // Outputs Mario
document.writeln(myFirstJSON["lastName"]);   // Outputs Rossi
document.writeln(myFirstJSON["age"]);        // Outputs 23
```

- Be aware that this is NOT an associative array, however it appears.
- If you attempt to loop through myFirstObject you will get in addition to the three properties above
  - any methods or prototypes assigned to the object
  - you're more than free to use this method of addressing JSON data
  - just treat it for what it is (**Object Properties**)
    - and not for what it is not (**Associative Array**)

Receiving JSON via AJAX

- There are three seperate ways to receive JSON data via AJAX.
  - Assignment
  - Callback
  - Parse

JSON Via Assignment

- There's no standard naming convention for these methods
  - however “assignment method” is a good descriptive name
  - The file *comming in* from the server creates a javascript
    - which will assign the JSON object to a variable
  - when the responseText from the server is passed through eval
    - someVar will be loaded with the JSON object
      - you can access it from there.

JSON Via Assignment

```
// example of what is received from the server.
var JSONFile = "someVar = { 'color' : 'blue' }";

// Execute the javascript code contained in JSONFile.
eval(JSONFile);

// It's amazing!
document.writeln(someVar.color); // Outputs 'blue'
```

- When the responseText from the server is passed through eval
  - someVar will be loaded with the JSON object
    - you can access it from there.

JSON Via Callback

- The second method calls a pre-defined function
  - passing the JSON data to that function as the first argument
- A good name for this method is the "callback method"
  - used extensively when dealing with third party JSON files
    - (IE, JSON data from domains you do not control).

```
function processData(incommingJSON) {
   document.writeln(incommingJSON.color); // Outputs 'blue'
}

// example of what is received from the server...
var JSONFile = "processData( { 'color' : 'blue' } )";

eval(JSONFile);
```

JSON Via Parse

- As JSON is just a string of text
  - it needs to be converted to an object before to make it useful.
  - Although this can be done in JavaScript with the eval() function
    - It is safer to use a JSON parser
    - You can download the JavaScript JSON parser at
      - http://www.json.org/json.js
      - Including this file on a web page allows you to take advantage of the JSON object

```
JSON.parse(strJSON) - converts a JSON string into a JavaScript object.
JSON.stringify(objJSON) - converts a JavaScript object into a JSON string.
```

JSON and PHP

- As of PHP 5.2.0, the JSON extension is bundled and compiled into PHP by default.
- Decoding JSON string is very simple with PHP
  - Only one line of code is needed to parse string into object.
  - Similary only one function is needed to encode PHP object or array into JSON string

```
<?php // decode JSON string to PHP object
$decoded = json_decode($_GET['json']); ?>
```

```
// create response object
$json = array();
$json['errorsNum'] = 2;
$json['error'] = array();
$json['error'][] = 'Wrong email!';
$json['error'][] = 'Wrong hobby!';

// encode array $json to JSON string
$encoded = json_encode($json);
```

AJAX database example

- The HTML page contains a link to an external JavaScript, an HTML form, and a div element:

```
<html>
<head>
<script type="text/javascript" src="selectuser.js"></script>
</head>
<body>

<form>
Select a User:
<select name="users" onchange="showUser(this.value)">
<option value="1">Peter Griffin</option>
<option value="2">Lois Griffin</option>
<option value="3">Glenn Quagmire</option>
<option value="4">Joseph Swanson</option>
</select>
</form>
<br />
<div id="txtHint"><b>Person info will be listed here.</b></div>

</body>
</html>
```

AJAX database example - The JavaScript

```
var xmlhttp;
function showUser(str) {
   xmlhttp=GetXmlHttpObject();
   if (xmlhttp==null) {
     alert ("Browser does not support HTTP Request");
     return;
   }
   var url="getuser.php";
   url=url+"?q="+str;
   url=url+"&sid="+Math.random();
   xmlhttp.onreadystatechange=stateChanged;
   xmlhttp.open("GET",url,true);
   xmlhttp.send(null);
}
function stateChanged() {
   if (xmlhttp.readyState==4) {
      document.getElementById("txtHint").innerHTML=xmlhttp.responseText;
   }
}
function GetXmlHttpObject() {
   if (window.XMLHttpRequest) {
     // code for IE7+, Firefox, Chrome, Opera, Safari
     return new XMLHttpRequest();
   }
   if (window.ActiveXObject) {
     // code for IE6, IE5
     return new ActiveXObject("Microsoft.XMLHTTP");
   }
   return null;
}
```

AJAX database example - The PHP Page

```
<?php
$q=$_GET["q"];

$con = mysql_connect('localhost', 'peter', 'abc123');
if (!$con) {
  die('Could not connect: ' . mysql_error());
}

mysql_select_db("ajax_demo", $con);

$sql="SELECT * FROM user WHERE id = '".$q."'";

$result = mysql_query($sql);

echo "<table border='1'>
<tr>
<th>Firstname</th>
<th>Lastname</th>
<th>Age</th>
<th>Hometown</th>
<th>Job</th>
</tr>";

while($row = mysql_fetch_array($result)) {
  echo "<tr>";
  echo "<td>" . $row['FirstName'] . "</td>";
  echo "<td>" . $row['LastName'] . "</td>";
  echo "<td>" . $row['Age'] . "</td>";
  echo "<td>" . $row['Hometown'] . "</td>";
  echo "<td>" . $row['Job'] . "</td>";
  echo "</tr>";
}
echo "</table>";

mysql_close($con);
?>
```