Computer Science E-75: Building Dynamic Websites
Harvard Extension School
Spring 2008

Section 6: Ajax (Asynchronous Javascript and XML)

Ajax is a type of programming – not really a separate language. It introduces a new way of using existing standards (specifically javascript and xml). By using the JavaScript XMLHttpRequest object, data (small bits of information) can be retrieved from a web server without reloading the entire page. This makes for really dynamically created websites, and is the method of choice for such applications as Google Maps.

Simple Ajax: A Body Mass Index (BMI) Calculator

Let's use Ajax to make a simple form that, when the submit button is pressed, retrieves data from the server and displays it without reloading the entire page. For this example, let's implement a BMI Calculator. A person's BMI is a somewhat accurate indication of how healthy someone is (a person with BMI of 18.5-24.9 is considered healthy). For more information, see http://www.whathealth.com/bmi/formula.html. The formula for BMI is as follows:

$$\text{BMI} = \frac{\text{(Weight in kilograms)}}{\text{(Height in meters) X (Height in meters)}}$$

We will first write a form using XHTML to get user input. This will look something like the following:

```
<form onsubmit="calculate(); return false;">
Weight (kg): <input id="weight" type="text" /><br />
Height (m): <input id="height" type="text" />
<br />
<input type="submit" value="Calculate BMI!" />
</form>
<br /><br />
Your BMI is: <span id="bmi"></span>
```

Notice that the span tags don't have anything between them. They are serving as placeholders right now. We will need to put our results back somewhere into the page, and an easy way for determining where is to put an id on an empty div or span.

You will also notice that the form tag attributes have no methods or actions. Instead, we have a JavaScript event handler onsubmit. When the submit button is pressed, the function calculate() will be called, and a false will be returned, stopping the data from being sent to the server the way we typically have in the past. The "magic" of Ajax is hidden within our function calculate(). Let's define calculate now. Within the head tags and the CDATA markers, we will enter the following:

```
var xhr = null;          // notice we declare a global variable xhr!

function calculate()
        {
            // instantiate XMLHttpRequest object
            try
            {
                xhr = new XMLHttpRequest();
            }
            catch (e)
            {
                xhr = new ActiveXObject("Microsoft.XMLHTTP");
            }

            // handle old browsers
            if (xhr == null)
            {
                alert("Ajax not supported by your browser!");
                return;
            }

            // construct URL
                   var url = "bmi.php?weight=" +
                        document.getElementById("weight").value
                        + "&height=" +
                        document.getElementById("height").value;

            // get quote
            xhr.onreadystatechange = handler;
            xhr.open("GET", url, true);
            xhr.send(null);
        }
```

First, we are trying to instantiate our XMLHttpRequest object, which makes this all possible. This is done multiple ways, depending on browsers, so in order to make sure it works, we will use the try-catch statement in JavaScript. Basically, this means we will try the "try" part, and if that fails, we will perform the "catch" part. A truly cross-browser implementation of this actually will have about four try-catches in order to make sure it works on every browser, but this is enough for a simple demonstration.

We also handle browsers that don't support either of the two previous assignments with the next check, if xhr comes back null.

From here, we construct a url that we will send (to pass the information in our form to a script called bmi.php that will take in the GET variables and return a BMI). We use JavaScript to get the values within the elements with id's "weight" and "height" and manually pass in the GET variables through the url.

Then, we perform three function calls on the xhr (XMLHttpRequest object) – onreadystatechange, open, and send (description of each on following page).

onreadstatechange – allows us to set an event handler (a function) that will respond to asynchronous events (such as the retrieval of data from a server).

open("GET", url, true); – Describes the method of sending information (GET), the url to which the information is sent (url) and that the data should be sent asynchronously (true). To send data asynchronously means that the data can be sent, and the website doesn't have to wait for a response before doing anything else. The user can still operate the webpage while the request is being made. For more documentation on this function, see http://msdn2.microsoft.com/en-us/library/ms536648(VS.85).aspx.

send(null); – Initiates the request. Documentation can be found at http://msdn2.microsoft.com/en-us/library/ms536736(VS.85).aspx.

Two things have left to be done. We have to write our script bmi.php to return a user's bmi. We also have to define the event handler function that is called when the information from the server is received. Let's define the handler first (this goes within the script tags in the head tags as well).

```
function handler()
{
	// only handle loaded requests
	if (xhr.readyState == 4)
	{
		if (xhr.status == 200)
		document.getElementById("bmi").innerHTML = xhr.responseText;

		else
		alert("Error with Ajax call!");
	}
}
```

This handler first checks whether the request has been loaded (which means the object's readyState is 4). If this is true, and the status of the XMLHttpRequest object is 200 (which means no errors), then the received data (xhr.responseText) is inserted within the element with id "bmi" (the span tags we created earlier) via the innerHtml property.

Other types of data can be received:
responseBody - returned data in binary format (not supported in all browsers)
responseText - returned data as a string
responseXML - if what is returned is xml, and you want a dom to get returned, use this
(a parsed version of responseText)

Now let's write a small script in bmi.php to calculate our bmi.

```
<?
      $height = $_GET[‘height’];
      $weight = $_GET[‘weight’];
      $bmi = $weight / ($height * $height);
      echo $bmi;
?>
```

Recognize that this script does not implement any server side validation of user input. A good implementation would include checks for non-numeric, 0, or negative values. This will simply echo the result of the bmi calculation, which will then be returned and “handled” by the handler.

To review, the five main things we needed for this example include:

1. A form for user input (form tags)
2. A placeholder for the returned data to reside (span tags)
3. A function to be called when submit is pressed (calculate())
4. An event handler to be called when information is received (handler())
5. A script to process the information (bmi.php)

Returning and Parsing XML Using Ajax (BMI Calculator)

This example will build upon our BMI Calculator, and return an xml formatted object that can be traversed using JavaScript. Let’s say that we want our output to look like this:

```
Your height is: <height goes here>
Your weight is: <weight goes here>
Your BMI is: <BMI goes here>
```

To do so, we can have Ajax return us an xml object that will have a structure like:

```
<calculator>
      <height>##</height>
      <weight>##</weight>
      <bmi>##</bmi>
</calculator>
```

With the ##’s standing in for numbers dependent upon user input. To achieve this, we only need change/edit four things.

1. We must create a new script (bmi2.php) that returns xml formatted text
2. We must reference that new script in our calculate function
3. We must change the placeholders in the file where the received data will go
4. We must change the handler to deal with the xml

New Script! (bmi2.php)

```
<?
      // set MIME type
      header("Content-type: text/xml");

      // print xml
      print(“<calculator>”);
      $height = $_GET[‘height’];
      $weight = $_GET[‘weight’];
      $bmi = $weight / ($height * $height);
           print("<height>{$height}</height>");
           print("<weight>{$weight}</weight>");
           print("<bmi>{$bmi}</bmi>");
      print(“</calculator>”);
?>
```

Notice we set the MIME type for the type of information that the page is producing – xml. Then, we proceed to manually print out the xml in the way specified above.

Change Calculate()

This is a very simple fix, in the function calculate, when we set the var url to bmi.php?.... we now want to change that to bmi2.php?.... to access the new script.

Change the Placeholders

Instead of just one placeholder, like in the previous example:

```
Your BMI is: <span id=”bmi”></span>
```

We will put in three placeholders, for each of the entries we want to return:

```
Your height is: <span id=”returnheight”></span>
Your weight is: <span id=”returnweight”></span>
Your BMI is: <span id=”bmi”></span>
```

Update the Event Handler

The new handler function will look like this. Notice that a variable xml is being set to xhr.responseXML. This is the xml object returned by the script. We will reference the values within the tags of this object by using the getElementsByTagName() function, and then insert these values within the correct placeholder by the getElementsById() function.

xml.getElementsByTagName() - returns all elements whose tag name matches the argument. Could return multiple results.

xml.getElementsById() - returns the element with the id matching the argument. Should return only one result

```
function handler()
{
	// only handle requests in "loaded" state
	if (xhr.readyState == 4)
	{
		if (xhr.status == 200)
		{
			// get XML
			var xml = xhr.responseXML;

			// update height
			var heights = xml.getElementsByTagName("height");
			if (heights.length == 1)
			{
				var height = heights[0].firstChild.nodeValue;
				document.getElementById("returnheight").innerHTML =
				height;
			}

			// update weight
			var weights = xml.getElementsByTagName("weight");
			if (weights.length == 1)
			{
				var weight = weights[0].firstChild.nodeValue;
				document.getElementById("returnweight").innerHTML =
				weight;
			}

			// update bmi
			var bmis = xml.getElementsByTagName("bmi");
			if (bmis.length == 1)
			{
				var bmi = bmis[0].firstChild.nodeValue;
				document.getElementById("bmi").innerHTML = bmi;
			}
		}
		else
			alert("Error with Ajax call!");
	}
}
```

Notice that we are accessing the elements of our XML return object the same way we would XHTML with JavaScript. Notice also the syntax of firstChild.nodeValue. This specifies the value of the first child of such an element. To make this more clear, it may be advantageous to use a utility such as Firebug with Mozilla Firefox and see what is actually returned to your page. (Firebug extension: http://www.getfirebug.com/). Now we've returned an XML object using Ajax!

Returning a JSON (BMI Calculator)

A JSON, unlike simple text or xml, is a string version of a JavaScript object. This works closely with defined classes to simplify syntax. Let's take a look at how it works. To implement the retrieval of a JSON, we must modify three things in our current code.

1. Change the script to return a JSON (bmi3.php)
2. Change the script that is referenced from bmi2.php to bmi3.php
3. Change the event handler function to deal with JSONs

Write a Script that Outputs JSON (bmi3.php)

```
<?
    // defines a class with bmi information
    class BMI
    {
        public $height;
        public $weight;
        public $bmi;
    }

    // set MIME type
    header("Content-type: application/json");

    $bmi = new BMI();
    $height = $_GET['height'];
    $weight = $_GET['weight'];
    $bmi = $weight / ($height * $height);
    $bmi->height = $height;
    $bmi->weight = $weight;
    $bmi->bmi = $bmi;

    // output JSON
    print(json_encode($bmi));
?>
```

First we've defined a class with variables for height, weight, and bmi. Then the mimetype for the page is set to application/json, so that the information gets returned as such. Notice that the last function json_encode() takes the variable $bmi and automatically returns the JSON representation of it (a string version of a JavaScript object).

Change the Script Reference (from bmi2.php to bmi3.php)

Just.. Do it. See the above example if this doesn't make sense. Or post to the Phorum!

Now we need to handle this object.

Change the Event Handler function

```
function handler()
{
      // only handle requests in "loaded" state
      if (xhr.readyState == 4)
      {
            if (xhr.status == 200)
            {
                  // get json
                  var bmi = eval("(" + xhr.responseText + ")");

                  // update height
                  document.getElementById("returnheight").innerHTML =
                  bmi.height;

                  // update weight
                  document.getElementById("returnweight").innerHTML =
                  bmi.weight;

                  // update bmi
                  document.getElementById("bmi").innerHTML = bmi.bmi;
            }
            else
```

```
                alert("Error with Ajax call!");
        }
}
```

The JavaScript eval() evaluates our string and executes it as if it were script code, basically creating a JavaScript object out of the PHP class. Now, the various portions of the object can be accessed using the dot (.) operator. This simplifies syntax, so we need not worry about accessing firstChild or NodeValue.

Ajax is also used for many other useful applications in website development. Auto-complete fields, automatically updating pages, and the course website all use Ajax to increase the dynamic element of most web pages. There are many resources online available for developers who wish to use Ajax, including the YUI's and other sites such as http://script.aculo.us/ or http://www.ajaxdaddy.com/. Try Googling for Ajax tutorials and examples for more goodies.