Comp 519: Web Programming
Autumn 2014

Ajax

- Basic objects necessary
- Setting up the XMLHttpRequest object
- Making the call
- How the server responds
- Using the reply
- XML basics

The usual way we operate in the Web

- Typical browsing behavior consists of loading a web page, then selecting some action that we want to do, filling out a form, submitting the information, etc.
- We work in this sequential manner, requesting one page at a time, and have to wait for the server to respond, loading a whole new web page before we continue.
- This is also one of the limitations of web pages, where transmitting information between a client and server generally requires a new page to be loaded.

- JavaScript is one way to cut down on (some of) the client-server response time, by using it to verify form (or other) information *before* it's submitted to a server.

- One of the limitations of JavaScript is (or used to be) that there was no way to communicate directly with a web server.

- Another drawback to this usual sequential access method is that there are many situations where you load a new page that shares lots of the same parts as the old (consider the case where you have a "menu bar" on the top or side of the page that doesn't change from page to page).

Things change…

- We used to not have any alternative to this load/wait/respond method of web browsing.
- Ajax (sometimes written AJAX) is a means of using JavaScript to communicate with a web server without submitting a form or loading a new page.
- Ajax makes use of a built-in object, `XMLHttpRequest`, to perform this function.
- This object is not yet part of the DOM (Document Object Model) standard, but is supported (in different fashions) by Firefox, Internet Explorer, Safari, Opera, and other popular browsers.
- The term “Ajax” was coined in 2005, but the `XMLHttpRequest` object was first supported by Internet Explorer several years before this.

Ajax

- Ajax stands for “Asynchronous JavaScript and XML”.
- The word “asynchronous” means that the user isn’t left waiting for the server the respond to a request, but can continue using the web page.

- The typical method for using Ajax is the following:

1) A JavaScript creates an `XMLHttpRequest` object, initializes it with relevant information as necessary, and sends it to the server. The script (or web page) can continue after sending it to the server.
2) The server responds by sending the contents of a file or the output of a server side program (written, for example, in PHP).
3) When the response arrives from the server, a JavaScript function is triggered to act on the data supplied by the server.
4) This JavaScript response function typically refreshes the display using the DOM, avoiding the requirement to reload or refresh the entire page.

The Back End

- The part of the Ajax application that resides on the web server is referred to as the "back end".
- This back end could be simply a file that the server passes back to the client, which is then displayed for the user.
- Alternatively, the back end could be a program, written in PHP, Perl, Ruby, Python, C, or some other language that performs an operation and sends results back to the client browser.
- An `XMLHttpRequest` object can send information using the GET and POST methods to the server in the same way that an HTML form sends information.
- Recall from our previous discussions that the GET request encodes the information inside of the URL, while a POST request sends its data separately (and can contain more information than a GET request can).

Writing an Ajax application

- We have to write the “front end” of the application in JavaScript to initiate the request.
- The back end, as mentioned, processes the request and sends it’s response back to the client. The back end is typically a short program we write for performing some dedicated task. This could be scripted in any language that is capable of sending back communication to the browser, like PHP or Perl.
- We also need to write the JavaScript response function for processing the response and displaying any results (or alterations to the web page).

- The “x” in Ajax stands for XML, the extensible markup language. XML looks like HTML, which is no mistake as the latest versions of HTML are built upon XML. The back end could send data back in XML format and the JavaScript response function can process it using built-in functions for working with XML. The back end could also send plain text, HTML, or even data in the JavaScript format.

- We will discuss some of these methods for sending data back to the requesting client and how it can be processed.

The XMLHttpRequest object

- The `XMLHttpRequest` object is the backbone of every Ajax method. Each application requires the creation of one of these objects. So how do we do it?

- As with most things in web programming, this depends upon the web browser that the client is using because of the different ways in which the object has been implemented in the browsers.

- Firefox, Safari, Opera, and some other browsers can create one of these objects simply using the “new” keyword.

```
<script type="text/javascript">
          ajaxRequest = new XMLHttpRequest();

</script>
```

The XMLHttpRequest object (cont.)

- Microsoft Internet Explorer implements this object using its proprietary ActiveX technology. This requires a different syntax for creating the object (and can also depend upon the particular version of Internet Explorer being used).

- To handle different types of browsers, we use the

```
try { . . . } catch (error) { . . . }
```

  format. The “try” section attempts to execute some JavaScipt code. If an error occurs, the “catch” section is used to intervene before the error crashes the JavaScript (either to indicate an error has happened, or to attempt something else).

- To create one of these objects we can use a sequence of try. . . catch blocks, attempting different ways to create an `XMLHttpRequest` object.

The XMLHttpRequest object (cont.)

```
function getXMLHttpRequest()
/*   This function attempts to get an Ajax request object by trying
     a few different methods for different browsers.  */
{
   var request, err;
   try {
          request = new XMLHttpRequest();   // Firefox, Safari, Opera, etc.
       }
   catch(err) {
       try {              //  first attempt for Internet Explorer
          request = new ActiveXObject("MSXML2.XMLHttp.6.0");
           }
       catch (err) {
                      try {    //  second attempt for Internet Explorer
                      request = new ActiveXObject("MSXML2.XMLHttp.3.0");
                          }
                      catch (err) {
                           request = false;  // oops, can't create one!
                                  }
                    }
              }
   return request;
}
```

If this function doesn't return "false" then we were successful in creating an `XMLHttpRequest` object.

The XMLHttpRequest object (cont.)

- As with any object in JavaScript (and other programming languages), the XMLHttpRequest object contains various properties and methods.
- We list the most important of these properties and methods on the next pages.
- The main idea is that the properties are set after the object is created to specify information to be sent to the server, as well as how to handle the response received from the server. Some properties will be updated to hold status information about whether the request finished successfully.
- The methods are used to send the request to the server, and to monitor the progress of the request as it is executed (and to determine if it was completed successfully).

XMLHttpRequest object properties

Property	Description
readyState	An integer from 0. . .4. (0 means the call is uninitialized, 4 means that the call is complete.)
onreadystatechange	Determines the function called when the objects readyState changes.
responseText	Data returned from the server as a text string (read-only).
responseXML	Data returned from the server as an XML document object (read-only).
status	HTTP status code returned by the server
statusText	HTTP status phrase returned by the server

We use the `readyState` to determine when the request has been completed, and then check the `status` to see if it executed without an error. (We'll see how to do this shortly.)

XMLHttpRequest object methods

Method	Description
• `open('method', 'URL', asyn)`	Specifies the HTTP method to be used (GET or POST as a string, the target URL, and whether or not the request should be handled asynchronously (asyn should be true or false, if omitted, true is assumed).
• `send(content)`	Sends the data for a POST request and starts the request, if GET is used you should call send(null).
• `setRequestHeader('x','y')`	Sets a parameter and value pair x=y and assigns it to the header to be sent with the request.
• `getAllResponseHeaders()`	Returns all headers as a string.
• `getResponseHeader(x)`	Returns header x as a string.
• `abort()`	Stops the current operation.

The `open` object method is used to set up the request, and the `send` method starts the request by sending it to the server (with data for the server if the POST method is used).

A general skeleton for an Ajax application

```
<script type="text/javascript">
       // *****  include the getXMLHttpRequest function defined before
var ajaxRequest = getXMLHttpRequest();

if (ajaxRequest) {   //  if the object was created successfully

      ajaxRequest.onreadystatechange = ajaxResponse;
      ajaxRequest.open("GET", "search.php?query=Bob");
      ajaxRequest.send(null);
   }


function ajaxResponse()  //This gets called when the readyState changes.
{
   if (ajaxRequest.readyState != 4)  //  check to see if we're done
      {  return;  }
   else {
     if (ajaxRequest.status == 200) //  check to see if successful
          {   //  process server data here. . . }
     else {
       alert("Request failed: " + ajaxRequest.statusText);
          }
     }
}
</script>
```

A first example

- Here's an example to illustrate the ideas we've mentioned (inspired by an example in the book Ajax in 10 Minutes by Phil Ballard).
- The main idea is that we're going to get the time on the server and display it to the screen (and provide a button for a user to update this time). The point I want to demonstrate here is how to use Ajax to do this update without updating/refreshing the entire webpage.

- We use a (very) small PHP script to get the date from the server, and return it as a string as a response to the request. Here is the script:

```
<?php
echo date('H:i:s');
?>
```

- I saved this as the file "telltime.php".
- The HTML file and JavaScript code follows.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>
<title>Ajax Demonstration</title>
<style>
body {
   background-color: #CCCCCC;
   text-align: center;
     }
.displaybox {
   margin: auto;
   width: 150px;
   background-color: #FFFFFF;
   border: 2px solid #000000;
   padding: 10px;
   font: 1.5em normal verdana, helvetica, arial, sans-serif;
           }
</style>

<script type="text/javascript">
var ajaxRequest;

function getXMLHttpRequest()
/*   This function attempts to get an Ajax request object by trying
     a few different methods for different browsers.  */
{
      //  same code as before. . .
}
```

```
function ajaxResponse()  //This gets called when the readyState changes.
{
   if (ajaxRequest.readyState != 4)  //  check to see if we're done
      {  return;  }
   else {
     if (ajaxRequest.status == 200) //  check to see if successful
          {
                   document.getElementById("showtime").innerHTML =
                                  ajaxRequest.responseText; }
     else {
       alert("Request failed: " + ajaxRequest.statusText);
          }
     }
}


function getServerTime()   //  The main JavaScript for calling the update.
{
   ajaxRequest = getXMLHttpRequest();
   if (!ajaxRequest)  {
         document.getElementById("showtime").innerHTML = "Request error!";
         return;      }
   var myURL = "telltime.php";
   var myRand = parseInt(Math.random()*999999999999999);
   myURL = myURL + "?rand=" + myRand;
   ajaxRequest.onreadystatechange = ajaxResponse;
   ajaxRequest.open("GET", myURL);
   ajaxRequest.send(null);
}
</script>
</head>
```

```
<body onload="getServerTime();">
<h1>Ajax Demonstration</h1>

<h2>Getting the server time without refreshing the page</h2>

<form>
   <input type="button" value="Get Server Time" onclick="getServerTime();" />
</form>
<div id="showtime" class="displaybox"></div>

</body>
</html>
```

The main functionality is handled by the `getServerTime()` function in setting up and sending the `XMLHttpRequest` object, and the `ajaxResponse()` function to display the time.

view the output page

What's this business with the random numbers?

- Web browsers use caches to store copies of the web page. Depending upon how they are set up, a browser could use data from its cache instead of making a request to the web server.

- The whole point of Ajax is to make server requests and not to read data from the cache. To avoid this potential problem, we can add a parameter with a random string to the URL so that the browser won't be reading data from its cache to satisfy the request (as then it looks like a different request than previous ones).

- This is only necessary if the request method is GET, as POST requests don't use the cache. (This also seems to be more of an issue with Microsoft Internet Explorer than with other browsers.)

Sending text back the server

- The response stored in `XMLHttpRequest.responseText` from the server can be any text that JavaScript is capable of processing as a string.
- Thus, you can send back a simple text string as the first example did, or you could send a string with HTML tags embedded in it. You can process the string using JavaScript functions (to split it into substrings, add/delete parts of it, etc.).
- You could even send back a string that has JavaScript code it in and execute it using the JavaScript `eval()` method.
- Recall, however, that the `responseText` property is a read-only variable, so if you're going to alter it you must first copy it to another variable.

Example with HTML tag (Change the PHP script to insert HTML tags.)

Example using a table (As above, change the PHP script.)

The other PHP scripts for the time examples

- Here's the script with a simple HTML tag in it.

```
<?php
echo '<span style="color: red;">' . date('H:i:s') . "</span>";
?>
```

- The output with a table.

```
<?php
$tr = '<tr style="border: 2px solid;">';
$td = '<td style="border: 2px solid">';

$table = '<table style="border: 2px solid; margin: auto;">';
$table .= $tr . $td . date('j M Y') . '</td></tr>';
$table .= $tr . $td . date('H:i:s') . '</td></tr>';
$table .= '</table>';
echo $table;
?>
```

XML: a (very) brief intro

- XML, the eXtensible Markup Language, is used in many ways, the most relevant to us being the transfer of structured information.

- XML and HTML look similar in many ways and this is because both are based on SGML, the Standard Generalized Markup Language established by the International Organization for Standards (ISO).

- Like HTML, XML uses tags to denote information but is not limited to the types of tags that occur in HTML. Tags can be essentially anything a user likes and are used to define the type of data present in the document.

XML: a (very) brief intro (cont.)

- Here's an example:

```
<library>
    <book>
        <title>Programming PHP</title>
        <author>Rasmus Lerdorf</author>
        <author>Kevin Tatroe</author>
        <author>Peter MacIntyre</author>
        <chapter number="1">Introduction to PHP</chapter>
        <chapter number="2">Language Basics</chapter>
        . . .
        <pages>521</pages>
    </book>
  . . .
</library>
```

- See the other notes for some more details/examples.

view other notes

Accessing an XML document in JavaScript

- To use an XML document in JavaScript, you must create an object to hold it. This can be done in the following fashion:
- Non-Microsoft browsers:

```
<script>
    var myXMLDoc = document.implementation.createDocument("","",null);
    myXMLDoc.load("mydoc.xml");
    //  other code here
</script>
```

- Internet Explorer:

```
<script>
    var myXMLDoc = new ActiveXObject("Microsoft.XMLDOM");
    myXMLDoc.async="false";
    myXMLDoc.load("mydoc.xml");
    //  other code here
</script>
```

- Once we've created the object holding the XML document, we can then use JavaScript methods to examine it, extract data from it, etc.

The “time” example using XML

- The first change is to make a new PHP script that returns an XML document to the browser.

```
<?php
header('Content-Type: text/xml');
echo "<?xml version=\"1.0\" ?>\n";
echo "<clock><timenow>" . date('H:i:s') .  "</timenow></clock>";
?>
```

- After that change (and inserting the new script name into the HTML code), we need to alter the `ajaxResponse` function to parse the XML document. That new JavaScript function is given on the next page.

- Note that we need not explicitly create an object to hold the XML document, but that responseXML (as a property of XMLHttpRequest) is already such an object.

The new Ajax response function

```
function ajaxResponse()  //This gets called when the readyState changes.
{
   if (ajaxRequest.readyState != 4)  //  check to see if we're done
      {  return;  }
   else {
     if (ajaxRequest.status == 200) //  check to see if successful
          {   var timeValue =
                    ajaxRequest.responseXML.getElementsByTagName("timenow")[0];
               document.getElementById("showtime").innerHTML =
                      timeValue.childNodes[0].nodeValue; }
     else {
       alert("Request failed: " + ajaxRequest.statusText);
          }
     }
}
```

- This new response function uses a JavaScript method to access the XML DOM and retrieve the time string before inserting it into the output display box.

view the output page

A second example (live search)

- We'll build a "live search" function. When you typically use a form, you must submit the data to the server and wait for it to return the results. Here we want to consider a case where you get (partial) results *as you enter data into the input field*, and that these results are updated (almost) instantly.

- Note: This example has been adapted from the book JavaScript in 24 Hours by Michael Moncur.

- We use PHP again for the backend. First consider the case where the possible results are a list of names, stored in a PHP array. As you type data into the input field, it's matched against this list of names, and the (partial) matches are returned and displayed on screen.

- Later, we will see the same type of application, but using PHP to search through the names stored in a database.

The HTML layout (no JavaScript yet)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>
<title>Ajax Demonstration</title>
<style>
</style>

<script>
    //   The JavaScript front end will be in here.
</script>

<body>
<h1>Ajax Demonstration of Live Search</h1>
<p>
Search for: <input type="text" id="searchstring" />
</p>
<div id="results">
<ul id="list">
<li>Results will be displayed here.</li>
</ul>
</div>

<script type="text/javascript">   //  This sets up the event handler to start the
                                  //           search function.
// var obj=document.getElementById("searchstring");
// obj.onkeydown = startSearch;
</script>
</body>
</html>
```

view the output page

The PHP backend

```
<?php
header("Content-Type: text/xml");
$people = array( "Abraham Lincoln", "Martin Luther King", "Jimi Hendrix",
   "John Wayne", "John Carpenter", "Elizabeth Shue", "Benny Hill",
   "Lou Costello", "Bud Abbott", "Albert Einstein", "Rich Hall",
   "Anthony Soprano", "Michelle Rodriguez", "Carmen Miranda",
   "Sandra Bullock", "Moe Howard", "Ulysses S. Grant", "Stephen Fry",
   "Kurt Vonnegut", "Yosemite Sam", "Ed Norton", "George Armstrong Custer",
   "Alice Kramden", "Evangeline Lilly", "Harlan Ellison");

if(!$query) $query = $_GET['query'];
echo "<?xml version=\"1.0\"?>\n";
echo "<names>\n";
while (list($k,$v) = each ($people))
   {
      if (stristr ($v, $query))
         echo "<name>$v</name>\n";
   }
echo '</names>';
?>
```

- This PHP script takes the query that will be passed to it, then searches for (case insensitive) matches to the names in the array.
- It passes an XML document back to the calling function consisting of the names that it finds.

The JavaScript functions

- We obviously need the function for creating a new `XMLHttpRequest` object, which we will store in a global variable called “ajaxRequest”.

- The search will be handled by setting up a Timeout event, based on entering text in the input field (using the “onkeydown” attribute).

```
var t;  //  public variable for the timeout

function startSearch()
{
   if (t) window.clearTimeout(t);
   t = window.setTimeout("liveSearch()",200);
}
```

- The “liveSearch” function is the main calling routine, where we set up the XMLHttpRequest object, and make the call to the PHP script.

The JavaScript functions (cont.)

- Recall that we're making ajaxRequest a global variable in the script, so that as in the other example we can access it's properties in the callback function.

```
function liveSearch()
{
   ajaxRequest = getXMLHttpRequest();
   if (!ajaxRequest)  alert("Request error!");
   var myURL = "search.php";
   var query = document.getElementById("searchstring").value;
   myURL = myURL + "?query=" + query;
   ajaxRequest.onreadystatechange = ajaxResponse;
   ajaxRequest.open("GET", myURL);
   ajaxRequest.send(null);
}

function ajaxResponse()  //This gets called when the readyState changes.
{
   if (ajaxRequest.readyState != 4)  //  check to see if we're done
      {  return;  }
   else {
     if (ajaxRequest.status == 200) //  check to see if successful
          {   displaySearchResults();   }
     else {
       alert("Request failed: " + ajaxRequest.statusText);
          }
     }
}
```

The JavaScript functions (cont.)

```
function displaySearchResults()
//  This function will display the search results, and is the
//  callback function for the Ajax request.
{  var i, n, li, t;
   var ul = document.getElementById("list");
   var div = document.getElementById("results");

   div.removeChild(ul);  //  delete the old search results
   ul = document.createElement("UL");  //  create a new list container
   ul.id="list";

   //  get the results from the search request object
   var names=ajaxRequest.responseXML.getElementsByTagName("name");
   for (i = 0; i < names.length; i++)
       {
         li = document.createElement("LI");  //  create a new list element
         n = names[i].firstChild.nodeValue;
         t = document.createTextNode(n);
         li.appendChild(t);
         ul.appendChild(li);
       }
   if (names.length == 0) { // if no results are found, say so
         li = document.createElement("LI");
         li.appendChild(document.createTextNode("No results."));
         ul.appendChild(li);
      }
   div.appendChild(ul);  // display the new list
}
```

The finished product

- We add all of the functions (and the two global variables) to the header script section, uncomment the two lines in the other script section and we're good to go!

- The fact that the names are in a PHP script allows us to easily add more or delete some of them. If desired, you could have the "search array" in a separate PHP file and include it in the search routine script, allowing you to reuse the same code with many different lists.

view the output page

Using a database for the live search

- Instead of storing the names in an array, we could alter the PHP script to search through a mySQL database for matches.
- The JavaScript need not be changed (except for the name of the script to call).
- As before, the PHP script will return the names as an XML document, using methods for a case-insensitive search on the query string.
- A new PHP script is shown on the next page.

```php
<?php
header("Content-Type: text/xml");
echo "<?xml version=\"1.0\"?>\n";
echo "<names>\n";

if(!$query) $query = strtoupper($_GET['query']);

if($query != "")
  {
     include_once('db_access.php');
     $connection = mysql_connect($db_host, $db_username, $db_password);
     if(!$connection)
        {
           exit("Could not connect to the database: <br/>" . htmlspecialchars(mysql_error));
        }
     mysql_select_db($db_database);

     $select = "SELECT ";
     $column = "name ";
     $from = "FROM ";
     $tables = "people ";
     $where = "WHERE ";
     $condition = "upper(name) LIKE '%$query%'";

     $SQL_query = htmlentities($select . $column . $from . $tables . $where . $condition);
     $result = mysql_query($SQL_query);
     while ($row = mysql_fetch_row($result))
          {
           echo "<name>" . $row[0] . "</name>\n";
          }

     mysql_close($connection);
  }
echo "</names>";
?>
```

view the output page

Some cautions

- As with any JavaScript element, you can't (or shouldn't) rely upon a user's browser being able to execute JavaScript (some people turn it off on their browsers). (Of course, there are webpages that ignore this caution.)

- Debug carefully and on many different browsers. Ajax uses features that might not be present in all browsers or they may not operate in the same fashion.

- If you can, indicate to the user that "something is happening" or that something has changed on the page, otherwise they may not notice it.

- Ajax can possibly introduce strange behavior, like the "Back" button on the browser doesn't act like it did before (as with any dynamic website), or that if you use some "hidden" elements in your page (generated by Ajax), then they will likely not show up in a form that search engines will recognize.

- For the sake of us all (who will be looking at your pages), don't let "flashy behavior" be a substitute for actual content, or a well designed layout of your web pages.