CSC 498: Web Programming

Haidar Harmanani
Department of Computer Science and Mathematics
Lebanese American University
Byblos, 1401 2010 Lebanon

AJAX Defined

1. Asynchronous Javascript and XML
   - Term coined by Jesse James Garrett
   - www.adaptivepath.com
2. AJAX is not a new technology
   - Google calls their technique: Javascript
   - Also known as XMLHTTP technique
   - In use since at least 1997
3. A bundle of techniques
   - XML data interchange only
   - Passing Javascript methods to client
   - DHTML widgets
   - XML & XSLTs
4. Core techniques are centered around asynchronous communication to the server without a page refresh

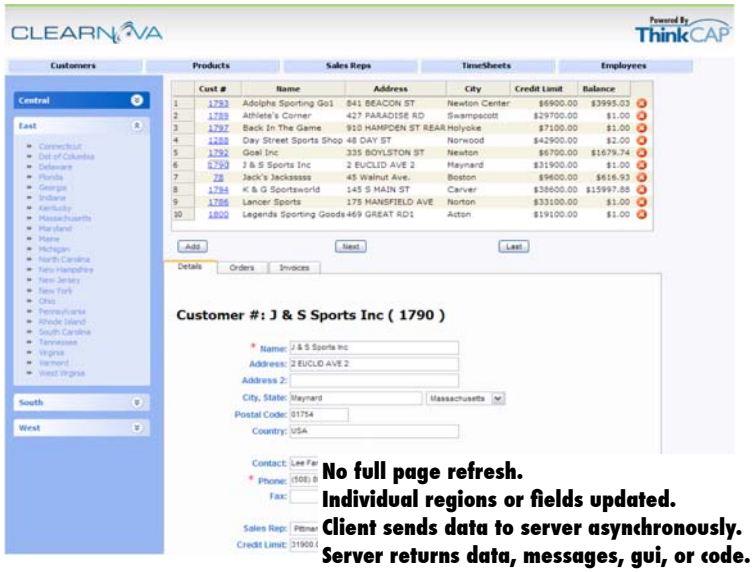
Underlying technologies

1. JavaScript for local processing
2. HTML
3. CSS and HTML for presenting
4. XML
   - XML is often used for receiving data from the server
   - Plain text can also be used, so XML is optional
5. DOM is used to access data inside the page or to access elements of XML file read on the server

So why is AJAX so hot—NOW?

1. Demand for richer applications is growing
   - Broadband means we can—and want to—do more
2. It has a name
   - Think LAMP—helped solidify the thoughts/techniques in people's minds
3. Recent Google applications have sparked people's imagination
   - Google gmail, Google suggests, Google Maps
4. People are thinking of building APPLICATIONS…not just sites
5. The Tipping Point
   - All of this has made rich Internet apps reach its tipping point—where adoption spreads rapidly and dramatically

AJAX Enabled Master Detail Form



No full page refresh.
Individual regions or fields updated.
Client sends data to server asynchronously.
Server returns data, messages, gui, or code.

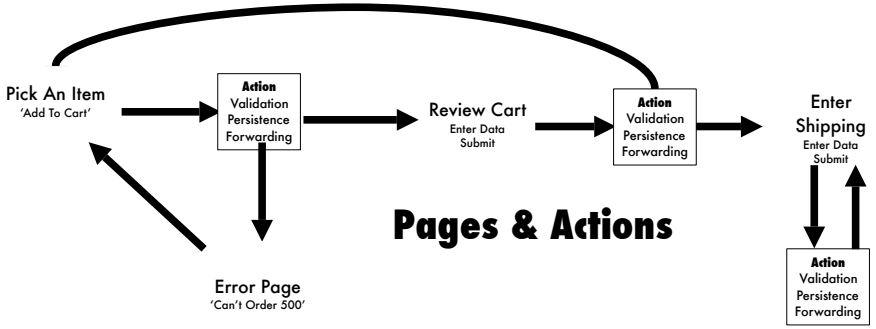
The usual way we operate in the Web

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

The usual way we operate in the Web

1. 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.
2. One of the limitations of JavaScript is (or used to be) that there was no way to communicate directly with a web server.
3. 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
   - The case where you have a "menu bar" on the top or side of the page that doesn't change from page to page

Traditional Web Applications…



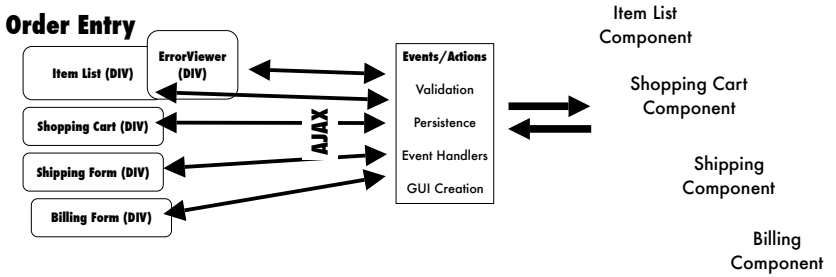
Unit of work is a page
Client code concerned with validation
Submits sent to actions
Actions perform work and then forward to next page
Page refresh for each submit

Things change…

1. Until recently, we didn't have any alternative to this load/wait/respond method of web browsing.
2. 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.
3. Ajax makes use of a built-in object, `XMLHttpRequest` , to perform this function.
4. 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.
5. The term "Ajax" was coined in 2005, but the `XMLHttpRequest` object was first supported by Internet Explorer several years before this.

AJAX Changes How Web Apps are Built

Components & Events



Unit of work is a component
Three-Tier Client/Server Model
Client code has validation, flow, layout, data interchange
No submit buttons—save buttons
Only parts of pages are updated at a time

AJAX Changes How Web Apps are Built

1. AJAX is not a new programming language
   - A technique for creating better, faster, and more interactive web applications.
2. With AJAX, JavaScript can communicate directly with the server, using the JavaScript `XMLHttpRequest` object.
   - JavaScript can trade data with a web server, without reloading the page.
3. AJAX uses asynchronous data transfer (HTTP requests) between the browser and the web server, allowing web pages to request small bits of information from the server instead of whole pages.
4. The AJAX technique makes Internet applications smaller, faster and more user-friendly

Some people don't like AJAX…

Two ways of talking to the server…

1. XMLHTTPRequest object
   - Allows for asynchronous GETs + POSTs to the server
   - Does not show the user anything—no status messages
   - Can have multiple XMLHTTPRequest active at one time
   - Allows you to specify a handler method for state changes
   - Handler notified when request is:
     - Initialized
     - Started
     - In the process of being returned
     - Completely finished
   - Originally only available for Microsoft IE

2. IFRAME
   - IFRAME is a "mini-browser" window in an HTML document
   - Can be hidden (0 width, 0 height)
   - IFRAME can call a URL
   - Javascript can read the contents of the IFRAME
   - User sees messages on status bar
   - Hears a click as server submits request
   - Much slower than XMLHTTPRequest

Ajax

1. Ajax stands for "Asynchronous JavaScript and XML".
2. 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.
3. Ajax uses a programming model with display and events
   - Events are user actions, they call functions associated to elements of the web page
4. 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.

Ajax

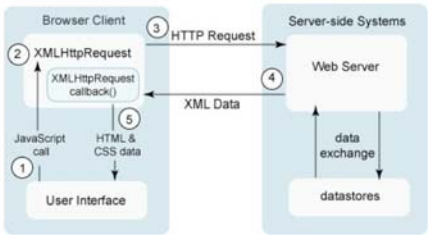
1. The typical method for using Ajax is the following:
   - A JavaScript creates an `XMLHttpRequest` object, initializes it with relevant information as necessary, and sends it to the server. The script (or web page) can continue after sending it to the server.
   - The server responds by sending the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

The Back End

1. The part of the Ajax application that resides on the web server is referred to as the "back end".
2. This back end could be simply a file that the server passes back to the client, which is then displayed for the user.
3. 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.
4. 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).

Ajax: How it works?

1. The user generates an event, resulting with a JavaScript call.
2. An XMLHttpRequest object is created and configured with a request parameter
3. The XMLHttpRequest object makes an asynchronous request to the web server. An object receives the request, processes it, and stores any data in the request to the data store.



4. The object that processed the request returns an XML document containing any updates that need to go to the client.
5. The XMLHttpRequest object receives the XML data, processes it, and updates the HTML DOM representing the page with the new data.

First Ajax Application

1. A standard HTML form with two text fields: username and time.
   - The username field will be filled in by the user and the time field will be filled in using AJAX

```
<html>
<body>
<form name="myForm">
Name: <input type="text" name="username" />
Time: <input type="text" name="time" />
</form>
</body>
</html>
```

AJAX Browser Support

1. The keystone of AJAX is the `XMLHttpRequest` object.
   - Different browsers use different methods to create the `XMLHttpRequest` object.
   - Internet Explorer uses an `ActiveXObject`, while other browsers use the built-in JavaScript object called `XMLHttpRequest`.
     - The XMLHttpRequest property is available on the window object in Internet Explorer 7
2. To create this object, and deal with different browsers, use a "try and catch" statement

The HTML form re-written

```
<html>
<body>
<script type="text/javascript">
function ajaxFunction()
{
var xmlHttp;
try
  {
  // Firefox, Opera 8.0+, Safari
  xmlHttp=new XMLHttpRequest();
  }
  catch (e)
  {
  // Internet Explorer
  try
    {
    xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
    }
  catch (e)
    {
    try
      {
      xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
      }
```

```
  catch (e)
     {
     alert("Your browser does not support
  AJAX!");
     return false;
```

More About the `XMLHttpRequest` Object

1. After a request to the server, we need a function that can receive the data that is returned by the server
   - The `onreadystatechange` property is an event handler that fires at each state change
   - Stores the function that will process the response from a server.
     - This is not a method, the function is stored in the property to be called automatically.
   - Sets the `onreadystatechange` property as follows:

```
xmlHttp.onreadystatechange=function()
{
// We are going to write some code here
}
```

The `readyState` Property

1. Add an If statement to the `onreadystatechange` function to test if our response is complete (this means that we can get our data)

```
xmlHttp.onreadystatechange=function()
{
   if(xmlHttp.readyState==4)
     {
     // Get the data from the server's response
     }
}
```

The `responseText` Property

1. The data sent back from the server can be retrieved with the `responseText` property

```
xmlHttp.onreadystatechange=function()
{
if(xmlHttp.readyState==4)
   {
   document.myForm.time.value=xmlHttp.responseText;
   }
}
```

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

- use the `readyState` to determine when the request has been completed, and then check the status to see if it executed without an error.

XMLHttpRequest Object Properties

1. onreadystatechange
   - Event handler that fires at each state change
   - You implement your own function that handles this
2. readyState – current status of request
   - 0 = uninitialized
   - 1 = loading
   - 2 = loaded
   - 3 = interactive (some data has been returned)
     - This is broken in IE right now
   - 4 = complete
3. status
   - HTTP Status returned from server: 200 = OK
4. responseText
   - String version of data returned from server
5. responseXML
   - XML DOM document of data returned
6. statusText

AJAX - Sending a Request to the Server

1. To send a request to the server, use the open() method and the send() method
   - The open() method takes three arguments.
     - The first argument defines which method to use when sending the request (GET or POST)
     - The second argument specifies the URL of the server-side script
     - The third argument specifies that the request should be handled asynchronously

```
xmlHttp.open("GET","time.php",true);
xmlHttp.send(null);
```

XMLHttpRequest Object: Methods

1. Allows to interact with the servers, thanks to its methods and attributes
   - open(mode, url, boolean)
     - open("method","URL")
     - open("method","URL", async, username, password)
       - Assigns destination URL, method, etc.
   - send(content)
     - Sends request including postable string or DOM object data
   - abort()
     - Terminates current request
   - getAllResponseHeaders()
     - Returns headers (labels + values) as a string
   - getResponseHeader("header")
     - Returns value of a given header
   - setRequestHeader("label","value")
     - Sets Request Headers before sending

XMLHttpRequest Object Methods

Property	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
`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 the complete list of response headers.
`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

AJAX - Sending a Request to the Server

1. Decide when the AJAX function should be executed
   - The function will run "behind the scenes" when the user types something in the username text

```
<form name="myForm">
   Name: <input type="text"
   onkeyup="ajaxFunction();" name="username" />
   Time: <input type="text" name="time" />
</form>
```

The HTML form re-written with Ajax

```
<html>
<body>

<script type="text/javascript">
function ajaxFunction()
{
var xmlHttp;
try
  {
  // Firefox, Opera 8.0+, Safari
  xmlHttp=new XMLHttpRequest();
  }
catch (e)
  {
  // Internet Explorer
  try
    {
    xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
    }
  catch (e)
    {
    try
      {
      xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
      }
```

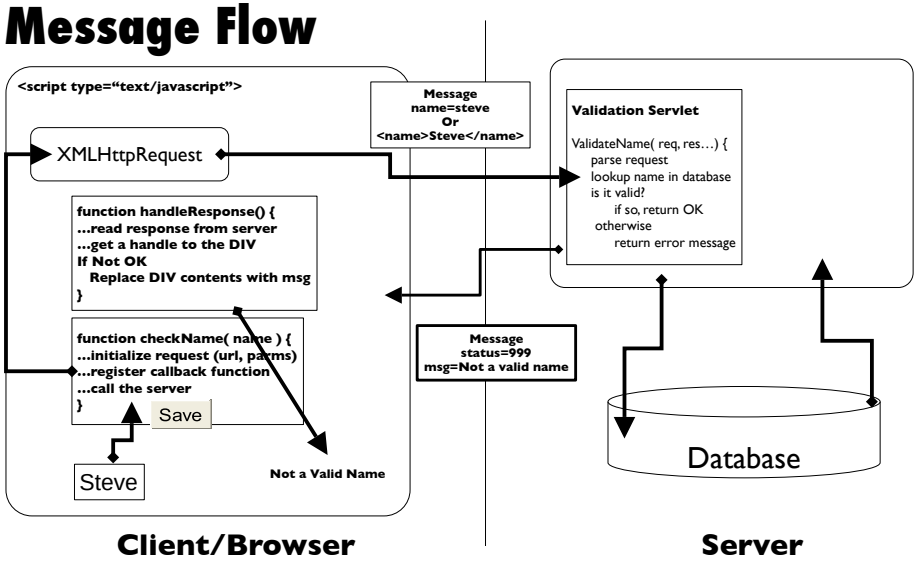
```
catch (e)
      {
      alert("Your browser does not support
  AJAX!");
      return false;
```

AJAX - The Server-Side ASP Script

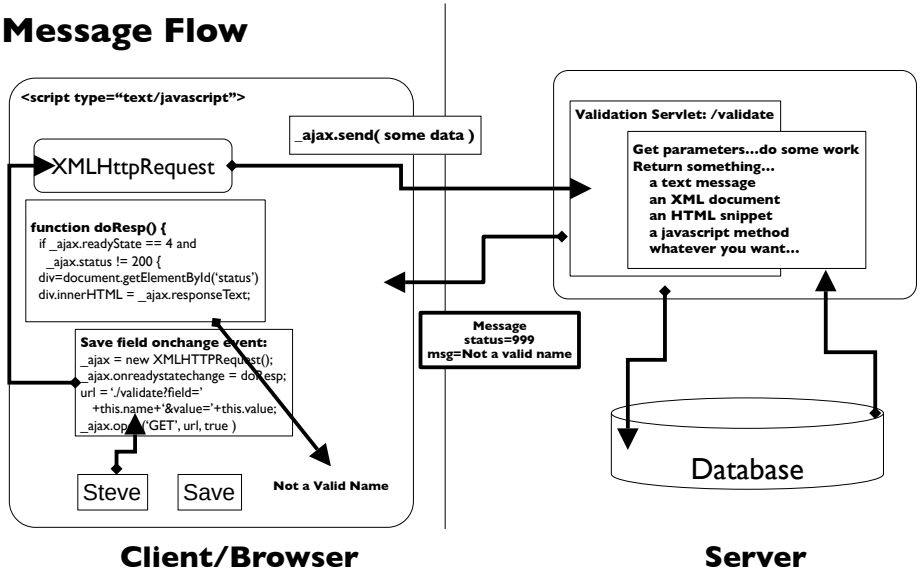
1. Create the script that displays the current server time
   - The `responseText` property (explained in the previous chapter) will store the data returned from the server. Here we want to send back the current time. The code in "time.asp" looks like this:

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

Message Flow



Message Flow



What is needed to build AJAX applications?

"Doing AJAX is rocket science, it's so hard."

Brian Goldfarb, Product Manager
Microsoft Web Platform and Tools Group

"Yea..have you seen the 20+ pages of documentation you need to follow to do autocomplete with Microsoft Atlas? No wonder they think its hard."

Ram Venkataraman, CTO, ClearNova

1. In some ways, AJAX development is harder than traditional web development.
   - Many more moving parts
     - more granular
     - events
   - JavaScript & DOM
   - CrossBrowser Issues
2. Tools & frameworks needed to make things easier

Writing an Ajax application

1. Let us now go over how we can write Ajax applications using a more detailed example

Writing an Ajax application

1. We have to write the "front end" of the application in JavaScript to initiate the request.
2. 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.
3. We also need to write the JavaScript response function for processing the response and displaying any results (or alterations to the web page).

Writing an Ajax application

1. The "x" in Ajax stands for XML, the extensible markup language.
   - 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.
2. We will discuss some of these methods for sending data back to the requesting client and how it can be processed.

The XMLHttpRequest object

1. The XMLHttpRequest object is the backbone of every Ajax method.
2. 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, IE7, 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.)

1. Clients prior to Internet Explorer 7 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).
2. As explained earlier, use `try { . . . } catch (error) { . . . }` in order to handle different types of browsers

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.)

1. As with any object in JavaScript (and other programming languages), the `XMLHttpRequest` object contains various properties and methods.
   - 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.
2. 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

- Use the `readyState` to determine when the request has been completed, and then check the status to see if it executed without an error.

XMLHttpRequest Object Methods

Property	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
`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')`	Sends the data for a POST request and starts the request, if GET is used you should call send(null)
`getAllResponseHeaders()`	Sets a parameter and value pair x=y and assigns it to the header to be sent with the request
`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

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>
```

Notes on Status

Number	Description
100	Continue
200	OK
201	Created
202	Accepted
206	Partial Content
401	Unauthorized
408	Request Timeout

A Second example

1. 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).
2. The main idea is that we're going to get the time on the server and display it to the screen
   - Provide a button for a user to update this time
   - Demonstrate how to use Ajax to do this update without updating/refreshing the entire webpage.
3. 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:

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

5. Save the script in a file "telltime.php".
6. 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?

1. Web browsers use caches to store copies of the web page.
   - Depending upon how they are set up, a browser could use data from it's cache instead of making a request to the web server.
2. 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).
3. This is only necessary if the request method is GET, as POST requests don't use the cache

Sending text back the server

1. The response stored in `XMLHttpRequest.responseText` from the server can be any text that JavaScript is capable of processing as a string.
2. 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.).

4. You could even send back a string that has JavaScript code it in and execute it using the JavaScript `eval()` method.

6. 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

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

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

2. 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;
?>
```

Accessing an XML document in JavaScript

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

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

4. Internet Explorer:

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

5. Once we've created the object holding the XML document, we can then use JavaScript

The "time" example using XML

1. 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>";
?>
```

2. 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.
3. 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("R
          }
     }
}
```

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 third example (live search)

1. We'll build a "live search" function.
2. When you typically use a form, you must submit the data to the server and wait for it to return the results.
3. 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.
4. Note: This example has been adapted from the book JavaScript in 24 Hours by Michael Moncur.

A third example (live search)

1. 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.
2. 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

1. We obviously need the function for creating a new XMLHttpRequest object, which we will store in a global variable called "ajaxRequest".
2. 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);
    }
```

3. 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.)

```
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);
          }
     }
}
```

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.

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

1. 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!
2. 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

1. Instead of storing the names in an array, we could alter the PHP script to search through a mySQL database for matches.
2. The JavaScript need not be changed (except for the name of the script to call).
3. As before, the PHP script will return the names as an XML document, using methods for a case-insensitive search on the query string.
4. A new PHP script is shown on the next page.
5. Important!!!
   - Remember that since you're now using PHP with mySQL, you need to access the main webpage via the CGI server.

```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

1. 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.)
2. 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.

Some cautions

1. 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.
2. 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.
3. 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.

What's Not to Like...

1. You have to program in Javascript
   - Need libraries + frameworks to make it easier
2. You have to program against the browser DOM
   - Differences between browser implementations
3. Lots of HTTP requests to server
   - Requests are smaller however
   - Total bandwidth may not go up
4. If you want client/server, why not use a real client/server system?
   - .NET can access Web Services on your Java boxes
   - Why not Flash/Flex?
   - Tons of good third-party rich client systems
5. Testing
   - Cross-browser quirks
   - New browser versions
6. No Back Button
   - Is this a bad thing?

Things To Consider…

1. How do you handle state + sessions?
   - The issues we had in the client/server world are back
   - Decision on where to put state information gets trickier
   - Can all be on the client so you can have truly stateless servers
   - Requires a lot more client code for state – server already has mechanism for clustered state synchronization
   - What happens if the server or network dies?
2. Navigation Issues
   - Refresh button can kill your app – open your application with minimal chrome
   - Back button is meaningless unless you code 'undo' functionality to it
   - Bookmarking doesn't work unless you manually save state + have a restore state method

Things To Consider…

1. Don't assume the server is going to respond instantly
   - Sending each keystroke for field validation is a dangerous idea
   - Block user judiciously—only when really needed
   - For something you expect to take a while—put a graphic or message up to show this might take a while
   - For a long running process with a known return length/time, use a progress bar