Ajax

Ronald J. Glotzbach
What is AJAX?

- Asynchronous JavaScript and XML
- Ajax is **not** a technology
- Ajax mixes well known programming techniques in an uncommon way
- Enables web builders to create more appealing user interfaces
What is Ajax? (cont.)

- Asynchronous:
  - The JavaScript makes the request to the server while you continue to interact with the webpage… without a page reload.

- Components:
  - Dynamic HTML
  - JavaScript
  - XML
Traditional Client-Server Interaction

- **CLIENT**
  - Web Browser
  - Web Page

- **SERVER**
  - Web Page
  - HTTP Request
  - HTTP Response
Ajax Client-Server Interaction
What is Ajax? (cont.)

- Ajax provides an extra layer of processing between the web page and the web server.

- This layer is often called:
  - Ajax Engine
  - Ajax Framework
Rich User Experience

- A rich user experience is similar to what a desktop application can give you. Fluid, quick, interactive components.

- Traditional web pages, while useful and effective, typically do not give you that rich user experience.
When you click a link or submit a form, you send an HTTP Request to the server.

Asynchronous

- Sending that HTTP Request **without** navigating and requesting a whole page

**XMLHttpRequest**

- This is a **JavaScript** object
- Makes an HTTP Request behind the scenes
XMLHTTPRequest (cont.)

- Can call a static XML file, which would not require any server-side code.

- Can call a server-side application for additional processing, such as database interaction.
When Should I use Ajax?

- When you only need to update part of a page.
  - Changing an image
  - Updating fields
  - Responding to users

Example:
- When creating a new user... checking to make sure the UserID entered is available while the user continues to enter data in the rest of the form
When Should I **not** use Ajax?

- While Ajax is an attractive alternative, there are cases where it detracts from the user experience instead of adding to it.
- When a form is completely filled out.
  - Click a submit button to send the request
- When the user will be navigating anyway
- Small sites may not sufficiently benefit considering the additional code required and increased complexity.
Most security issues surrounding Ajax are pre-existing.

Although, Ajax does use more client-side code that is visible to users:

- Coder must be cognizant not to place sensitive information there
  - i.e.: Make sure server-side code takes care of database connections, etc.
  - Make sure business logic resides on the server
  - Validate data on the server before using it
Client-Side Issues

- JavaScript is still implemented differently across platforms.
- The Document Object Model (DOM) is handled differently between browsers and between versions of the same browser.
- With the given complexity of Ajax, it is even more important now to test on multiple browsers.
Ajax won’t fix a bad design

Good web design still applies:

- Write for multiple browsers and validate your code
- Comment your code well so you can debug it later
- Use graphics small in size so they load quickly
- Make sure your choices of colors, backgrounds, font sizes, and styles follow proper web design techniques
Ajax

Code & Examples
Create an Instance

- In Mozilla, Opera, etc.:
  - var request = new XMLHttpRequest();

- However, in Internet Explorer:
  - var request = new ActiveXObject("Microsoft.XMLHTTP");

- However, some versions of Internet Explorer have different versions of the MSXML parser:
  - var request = new ActiveXObject("Msxml2.XMLHTTP");
Therefore, we need an example that will work *across* platforms and across multiple versions of the same web browser.
function getXMLHTTPRequest()
{
  var request = false;
  try
  {
    /* e.g. Firefox */
    request = new XMLHttpRequest();
  }
  catch(error1)
  {
    try
    {
      /* some versions IE */
      request = new ActiveXObject("Msxml2.XMLHTTP");
    }
    catch(error2)
    {
      try
      {
        /* some versions IE */
        request = new ActiveXObject("Microsoft.XMLHTTP");
      }
      catch(error3)
      {
        request = false;
      }
    }
  }

  return request;
}
Then, to generate an HTTP Request:

```javascript
var request = new getXMLHTTPHTTPRequest();
```

This calls the function (on the previous slide) and returns the appropriate XMLHttpRequest object.
Introducing try-catch

- Similar to placing an `@` in front of a PHP statement
- Similar to placing `OnError Resume Next` in ASP code
- A try-catch block allows the page to continue processing and **not** crash when an error occurs
  - Additionally, in the catch block, you can write customized code to deal with the error
# XMLHttpRequest Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onreadystatechange</td>
<td>Determines which event handler will be called when the object’s readyState property changes</td>
</tr>
</tbody>
</table>
| readyState       | Integer reporting the status of the request:  
|                  | 0 = uninitialized  
|                  | 1 = loading  
|                  | 2 = loaded  
|                  | 3 = interactive  
|                  | 4 = completed |
| responseText     | Data returned by the server in text string form |
| responseXML      | Data returned by the server expressed as a document object |
| status           | HTTP status code returned by server |
| statusText       | HTTP reason phrase returned by server |
## XMLHttpRequest Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abort()</td>
<td>Stops the current request</td>
</tr>
<tr>
<td>getAllResponseHeaders()</td>
<td>Returns all headers as a string</td>
</tr>
<tr>
<td>getResponseHeader(x)</td>
<td>Returns the value of header x as a string</td>
</tr>
<tr>
<td>open(‘post’, ‘URL’, ‘true’)</td>
<td>Specifies the HTTP method (for example, get or post), the target URL, and whether the request should be handled asynchronously (if yes, true --- if not, false)</td>
</tr>
<tr>
<td>setRequestHeader(‘x’, ‘y’)</td>
<td>Sets a name equal value pair, x=y, and assigns it to the header to be sent with the request</td>
</tr>
<tr>
<td>send(content)</td>
<td>Sends the request, optionally with post data</td>
</tr>
</tbody>
</table>
## HTTP Response Status Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>200 – OK</strong></td>
<td>The request succeeded</td>
</tr>
<tr>
<td><strong>204 – No Content</strong></td>
<td>The document contains no data</td>
</tr>
<tr>
<td><strong>301 – Moved Permanently</strong></td>
<td>The resource has permanently moved to a different URL</td>
</tr>
<tr>
<td><strong>401 – Not Authorized</strong></td>
<td>The request needs user authentication</td>
</tr>
<tr>
<td><strong>403 – Forbidden</strong></td>
<td>The server has refused to fulfill the request</td>
</tr>
<tr>
<td><strong>404 – Not Found</strong></td>
<td>The requested resource does not exist on the server</td>
</tr>
<tr>
<td><strong>408 – Request Timeout</strong></td>
<td>The client failed to send a request in the time allowed by the server</td>
</tr>
<tr>
<td><strong>500 – Server Error</strong></td>
<td>Due to a malfunctioning script, server configuration error, or something similar</td>
</tr>
</tbody>
</table>
// function from previous slide goes up here

var request = getXMLHTTPRequest();

function callAjax()
{
    // declare and initialize a variable
    var lastname = "Smith";

    // build the URL of the server script we wish to call
    var url = "serverscript.php?username=" + lastname;

    // use the XMLHTTPRequest object to open a server connection
    request.open("GET", url, true);

    // prepare a function responseAjax() to run when the response has arrived
    request.onreadystatechange = responseAjax;

    // and finally send the request
    request.send(null);
}
function responseAjax()
{
    // we are only interested in readyState of 4, i.e. "loaded"
    if(request.readyState == 4)
    {
        // if server HTTP response is "OK"
        if(request.status == 200)
        {
            alert("The server said: " + request.responseText);
        }
        else
        {
            // issue an error message for any other HTTP response
            alert("An error has occurred: " + request.statusText);
        }
    }
}
Example 1

- Get Server Time
  - Use PHP to generate XML output
  - Call the PHP using Ajax
  - Use JavaScript to parse and display the XML

- **Ajax Example 1**
Example 2

- Get Text From the Server
  - Use PHP to echo strings of text
  - Call the PHP using Ajax
  - Use JavaScript to display the text from the server

  - [Ajax Example 2](#)
Example 3

- Get Text From the Server
  - The same example as #2, but using ASP
  - Use ASP to Response. Write strings of text
  - Call the ASP using Ajax
  - Use JavaScript to display the text from the server

- Ajax Example 3
Now, imagine instead of just returning text in an echo or Response.Write… that some processing occurs on the server, perhaps verification of a username, and the result returned determines what text you display in the browser.
Asynchronous HTML and HTTP
You may see this referred to as different acronyms. This is essentially a subset of Ajax

- JAH
  - Just Asynchronous HTML
- HAJ
  - HTML and JavaScript (also called Dynamic HTML)
- AHAH
  - Asynchronous HTML and HTTP
- xmljs
  - XML and JavaScript (this is a popular term)
Example 4

- Retrieving content from a website
  - Use PHP to retrieve keywords from a website, if a keywords meta tag exists.
  - Type in a website into the form
  - Submit the form to have Ahah call the PHP
  - Write out the result from the PHP to the browser

- Example 4
Really, examples 2, 3, and 4 are considered AHAH because no XML was involved with those examples.
Now, let’s put the x in Ajax

The level of coding steps up a notch as more extensive JavaScript is used to navigate through the XML DOM
## XML DOM Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>childNodes</td>
<td>Array of child nodes</td>
</tr>
<tr>
<td>firstChild</td>
<td>The first child node</td>
</tr>
<tr>
<td>lastChild</td>
<td>The last child node</td>
</tr>
<tr>
<td>nodeName</td>
<td>Name of the node</td>
</tr>
<tr>
<td>nodeType</td>
<td>Type of node</td>
</tr>
<tr>
<td>nodeValue</td>
<td>Value contained in the node</td>
</tr>
<tr>
<td>nextSibling</td>
<td>Next node sharing the same parent</td>
</tr>
<tr>
<td>previousSibling</td>
<td>Previous node sharing the same parent</td>
</tr>
<tr>
<td>parentNode</td>
<td>Parent of this node</td>
</tr>
</tbody>
</table>
### XML DOM Node Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppendChild()</td>
<td>Add a new child node</td>
</tr>
<tr>
<td>HasChildNodes()</td>
<td>True if this node has children</td>
</tr>
<tr>
<td>RemoveChild()</td>
<td>Deletes a child node</td>
</tr>
</tbody>
</table>
## Document Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateAttribute()</td>
<td>Make a new attribute for an element</td>
</tr>
<tr>
<td>CreateElement()</td>
<td>Make a new document element</td>
</tr>
<tr>
<td>CreateTextNode()</td>
<td>Make a text item</td>
</tr>
<tr>
<td>GetElementsByTagName()</td>
<td>Create an array of tagnames</td>
</tr>
<tr>
<td>GetElementsById()</td>
<td>Find an element by its ID</td>
</tr>
</tbody>
</table>
Example 5

RSS Headline Reader

- Pass a URL to an RSS feed to PHP
- PHP reads the contents of the RSS feed using the cURL library and echo’s it out
- Use the DOM to navigate the output and display the contents

- [Ajax Example 5](#)
Example 6

- Read Directly from XML
  - Point Ajax to the xml file
  - Use the DOM to navigate the XML and display the contents

- Ajax Example 6
It is possible to create standalone applications in Ajax – meaning that no server is needed.

A standalone application should be able to be copied to a new location, such as your desktop, and function the same as if it were on a server.
Pulling JavaScript out as include files

- An added layer of ‘protection’, if you will. The typical user has to be savvy enough to decipher HTML, find the .js file, then navigate to that .js file in the browser explicitly in order to view it.

  - This is still **NOT** secure
Example 7

- Example 6 with JavaScript pulled out and placed in .js files
- Script include statements used within the html

Example 7
Web Services & REST

- **Web Service**
  - A server-side application that you typically interface with using XML and HTTP. It is not a website – it is an application that you send a request to, the web service processes the request, and sends you back a result.

- **REST**
  - Representational State Transfer
REST is sometimes referred to as:
- XML over HTTP
- XML/HTTP

Two main principals of REST:
- Resources are represented by URLs
  - Each web service has a unique URL
- Operations are carried out via HTTP
  - Get, Post, Put, Delete
REST Web Service

- An example call:
  - http://del.icio.us/api/[user]/bookmarks/?start=1&end=2

- Another example call:
  - http://xml.amazon.com/onca/xml3?t=[id]&dev-t=[token]&ManufacturerSearch=Sams&mode=books&sort=+salesrank&offer=all&type=lite&page=1&f=xml
Simple Object Access Protocol

- An XML-based messaging protocol

- Contains:
  - envelope (defines doc as a SOAP request)
  - Body (info about the call & expected responses)
  - optional header (supplementary info)
  - optional fault (supplementary info)
SOAP Example Request

```xml
<?xml version="1.0"?>

<soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

    <soap:Body xmlns:m="http://www.example.org/stock">

        <m:GetStockPrice>
            <m:StockName>IBM</m:StockName>
        </m:GetStockPrice>

    </soap:Body>

</soap:Envelope>
```
SOAP Example Response

HTTP/1.1 200 OK
Content-Type: application/soap+xml;
charset=utf-8 Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

    <soap:Body xmlns:m="http://www.example.org/stock">

        <m:GetStockPriceResponse>
            <m:Price>34.5</m:Price>
        </m:GetStockPriceResponse>

    </soap:Body>

</soap:Envelope>
Ajax Issues

- The Back Button
  - Every browser has one
  - What happens when it’s clicked?
    - Additional programming may be required for you to track your own state – and you may have to persuade users not to use the back button…
Ajax Issues (cont.)

- Bookmarking
  - Try using some of the Ajax examples and then bookmarking the page at a specific point.
  - When you navigate back to your bookmark, the state of the page is *not* maintained.
Ajax Issues (cont.)

- **Spiders**
  - Ajax does not typically return the most accurate information for search engine spiders
    - Ajax has interactive content whereas many spiders search for static content
  - You may need to create a static site map or other means to allow search engine spiders to properly index your site.
XML Object-oriented Application Development (XOAD)

XOAD is an Ajax toolkit written in PHP

- www.xoad.org
- http://sourceforge.net/projects/xoad
- http://forums.xoad.org/
Ajax

The End