AJAX

AJAX (originally an acronym for Asynchronous JavaScript and XML) allows a client-side JavaScript program to exchange data with a webserver. Typically the data sent will be in the form of an XML-file, although AJAX can be used to send text files as well.

Using AJAX, our client-side program will send a standard HTTP request to our webserver. Our program will setup a callback function which will be called when the webserver has responded with the data requested. Once received our callback function will process the data, typically modifying the webpage using the standard dynamic content techniques described in the previous handout.

XMLHttpRequest

The XMLHttpRequest object is the key to using AJAX. This object was first defined by Microsoft for use in Internet Explorer. It has since been adapted by other web browsers, and a proposed standard version is being drafted by the W3C.

Creating an XMLHttpRequest Object

In order to use AJAX, we'll need to create an XMLHttpRequest object. Here's the code for this:

```
requestObj = new XMLHttpRequest();
```

Setting Up a Request using XMLHttpRequest

Once we've created our XMLHttpRequest object, we're ready to put together a request for our webserver. We can use an HTTP GET, POST, or HEAD request (additional HTTP requests may be supported depending on the browser).

We'll need to take several steps in order to get our request ready. First, we tell the XMLHttpRequest object, which of our JavaScript functions to use as a callback when we receive a response back from the webserver. Next we put together our actual request to the webserver. Finally we actually send the request to the webserver.

Suppose, for example, we have a function named handleResponse that we want to execute when the webserver sends our data. We would assign the handler function to our XMLHttpRequest object as follows:

```
requestObj.onreadystatechange = handleResponse;
```

Newer web browsers allow us to specify a handler for specifically when the webpage loads using an attribute called onload. Unfortunately it's not supported by Internet Explorer, so I will stick to the older method here.

We'll take a look in a moment what our callback function actually needs to do when it executes.

Next we setup our request to the webserver using the XMLHttpRequest object's open and send methods. The open method takes three parameters—first, the type of request (e.g., GET, POST, HEAD),[1] second the actual URL requested, and third a boolean telling the system whether or not the call is asynchronous (that is whether the system should wait until the data is received or should continue regular operation until the data is received). The send method is used to send information in the request body as associated with a POST request. If you aren't using a POST call send with a null parameter.

Here is an example requesting the file info.xml, were we assume info.xml is located in the same directory as the current HTML file (the one whose JavaScript we are executing):

```
requestObj.open('GET', 'info.xml', true);
requestObj.send(null);
```

The URL requested can be a relative or absolute HTTP reference. Here we are requesting a specific file on the Stanford webserver:

```
requestObj.open('GET', 'http://www.stanford.edu/dept/cs/info.xml', true);
requestObj.send(null);
```

Warning: The URL requested must be on the same webserver as the webpage making the request. Accessing a "3rd party domain" using the XMLHttpRequest object is illegal (unless your user modifies the browser security settings).

There have been some attempts to setup way to get around this, but at this point they aren't standardized yet. For more information check the W3C's "Cross-Origin Resource Sharing" draft and Microsoft's XDomainRequest object.

Note: While we can create a synchronous HTTP request using the third parameter of the open call, this is not recommended. The user will not be able to interact with your web page while waiting for the web server to return its data.

Responding to Server Data

Your callback function will get called when the status of your request changes. There are five different possible statuses—the W3C and older Microsoft web browsers use different statuses. Here are the W3C statuses.

0. Uninitialized. This is the initial status of your request.
1. Open. This is the status of your request after you call the open method.
2. Sent. Your request has been sent, but no information from the web server has been received.
3. Receiving. Your computer is receiving data, the header information is available for processing but additional data (e.g., the actual file) is still being transferred.
4. Loaded. The transfer has been completed.

[1] For those of you unfamiliar with HTTP, we can send several different types of requests to a webserver. A GET request as the name implies requests a file from the webserver. A POST sends information to the webserver and receives a file in response. A HEAD is the same as a GET, except the actual file isn't sent, instead only the header accompanying a regular GET response is sent. This header contains information about the file (such as modification date and size).

Here are the older IE statuses.

0. Unintialized. This is the initial status of your request.

1. Loading. This is the status of your request after you call the send method.

2. Loaded. Your request has been sent and is awaiting processing on the webserver.

3. Interactive. The webserver is processing your request.

4. Completed. The transfer has been completed.

For most purposes, we are only concerned with state 4. As both IE and W3C agree that state 4 indicates that all data transfer has been completed, we can generally write a unified callback function. Most callback functions will look something like this:

```
function handleResponse() {
    if (requestObj.readyState == 4) {
        … // do your standard processing here
    }
}
```

where we only process if the XMLHttpRequest object's status indicates that we have received all the information back from the webserver.

Once the data has been received, check the status of the HTTP request using the XMLHttpRequest's status property. The status is the standard HTTP status number, such as 404 "File Not Found" or 500 "Internal Server Error". The status you want is 200 "Ok".

```
function handleResponse() {
    if (requestObj.readyState == 4) {
        if (requestObj.status == 200) {
            … // do your standard processing here
        } else
            alert("Problem Response from Server: " + requestObj.status);
    }
}
```

Processing Information

So, once the transfer has been completed, how do we get a hold of the information? Our actual data is in one of two places. If we've requested an XML document, we can access its document object model in the XMLHttpRequest object's responseXML property. If the file sent is a text file, it can be found in the object's responseText property.

If the object returns a responseXML, the object will be in the form of a DOM tree. You can use the functions we discussed in the Dynamic Content lecture in order to access the nodes in the tree. For example, the following code retrieves an array of all the elements in the XML file with tag name "title":

```
requestObj.responseXML.getElementsByTagName("title");
```

Caveats

- First to reiterate, unless your user has changed the browser security settings, you cannot use XMLHttpRequest to access a URL which is not from the same webserver as your webpage.

- Some versions of Firefox may require the following line added before the send in order to force the web browser to view the data received as XML:

  ```
  requestObj.overrideMimeType('text/xml');
  ```

  Unfortunately this line will make IE crash (IE's XMLHttpRequest doesn't support this method).

- The GET HTTP method is officially designated as "idempotent". This means that the value it returns should not change no matter how many times they are called. Because the value returned should not change, IE will generally cache the result. If you call GET multiple times during the same session, with the same arguments, after the first request, IE will simply go to the disk cache and retrieve the previous result.

Supporting IE6

The XMLHttpRequest object as previously describes should work in any modern web browser including IE7 and newer. Creating the XMLHttpRequest object in earlier versions of Internet Explorer is a bit different. We need to use Microsoft's ActiveX technology to create the object. Here's the code:

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

We can combine the two methods as follows (code taken from Mozilla's "Getting Started with AJAX" webpage):

```
if (window.XMLHttpRequest)
    requestObj = new XMLHttpRequest();
else if (window.ActiveXObject)
    requestObj = new ActiveXObject("Microsoft.XMLHTTP");
```

Here we first test to make sure the web browser knows what an XMLHttpRequest object is, if it does, we create it. Otherwise we check to see if it knows what an ActiveXObject is, if it does, we use ActiveX to create our XMLHttpRequest object.

Additional Information

The current W3C draft specifies a timeout property on the XMLHttpRequest object. Setting this property will tell the browser that we are only willing to wait for the given number of milliseconds before terminating the request.

A second property ontimeout specifies a function to call if the timeout occurs.

These properties are relatively late additions to the W3C draft, so if you plan to use them, test thoroughly on whatever web browsers your website is planning to support.