Department of
Computer Science and Engineering

Title: Asynchronous JavaScript and XML (AJAX)

Web Programming Lab

CSE 301

Green University of Bangladesh

1 Objective(s)

- To be familiar with AJAX.
- To be familiar with Asynchronous and Synchronous web requests.

2 Introduction

AJAX means Asynchronous JavaScript And XML. AJAX is not a programming language. AJAX just uses a combination of:

- A browser built-in XMLHttpRequest object (to request data from a web server)
- JavaScript and HTML DOM (to display or use the data)

AJAX is a misleading name. AJAX applications might use XML to transport data, but it is equally common to transport data as plain text or JSON text. AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

Modern Browsers can use Fetch API instead of the XMLHttpRequest Object. The Fetch API interface allows web browser to make HTTP requests to web servers. If you use the XMLHttpRequest Object, Fetch can do the same in a simpler way.

3 XMLHttpRequest Object

All modern browsers support the XMLHttpRequest object. The XMLHttpRequest object can be used to exchange data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

3.1 XMLHttpRequest Object Methods

Method	Description
new XMLHttpRequest()	Creates a new XMLHttpRequest object
abort()	Cancels the current request
getAllResponseHeaders()	Returns header information
getResponseHeader()	Returns specific header information
open(method, url, async, user, psw)	Specifies the request
send()	Sends the request to the server Used for GET requests
send(string)	Sends the request to the server. Used for POST requests
setRequestHeader()	Adds a label/value pair to the header to be sent

3.2 XMLHttpRequest Object Properties

Property	Description
onload	Defines a function to be called when the request is recieved (loaded)
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest
responseText	Returns the response data as a string
responseXML	Returns the response data as XML data
status	Returns the status-number of a request
statusText	Returns the status-text (e.g. "OK" or "Not Found")

3.2.1 The onload Property

With the XMLHttpRequest object you can define a callback function to be executed when the request receives an answer. The function is defined in the onload property of the XMLHttpRequest object:

```
xhttp.onload = function() {
  document.getElementById("demo").innerHTML = this.responseText;
}
xhttp.open("GET", "ajax_info.txt");
xhttp.send();
```

3.2.2 The onreadystatechange Property

The readyState property holds the status of the XMLHttpRequest. The onreadystatechange property defines a callback function to be executed when the readyState changes. The status property and the statusText properties hold the status of the XMLHttpRequest object.

onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest (0: request not initialized, 1: server connection established, 2: request received, 3: processing request, 4: request finished and response is ready)
status	Returns the status-number of a request (200: "OK", 403: "Forbidden", 404: "Page not found")
statusText	Returns the status-text (e.g. "OK" or "Not Found")

The onreadystatechange function is called every time the readyState changes.
When readyState is 4 and status is 200, the response is ready:

```
function loadDoc() {
  const xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML =
      this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt");
  xhttp.send();
}
```

The onreadystatechange event is triggered four times (1-4), one time for each change in the readyState.

3.3 Create an XMLHttpRequest Object

All modern browsers (Chrome, Firefox, IE, Edge, Safari, Opera) have a built-in XMLHttpRequest object. Syntax for creating an XMLHttpRequest object:

```
variable = new XMLHttpRequest();
```

3.4 Define a Callback Function

A callback function is a function passed as a parameter to another function. In this case, the callback function should contain the code to execute when the response is ready.

```
xhttp.onload = function() {
  // What to do when the response is ready
}
```

3.4.1 Multiple Callback Functions

If you have more than one AJAX task in a website, you should create one function for executing the XMLHttpRequest object, and one callback function for each AJAX task. The function call should contain the URL and what function to call when the response is ready.

```
loadDoc("url-1", myFunction1);

loadDoc("url-2", myFunction2);

function loadDoc(url, cFunction) {
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {cFunction(this);}
  xhttp.open("GET", url);
  xhttp.send();
}

function myFunction1(xhttp) {
  // action goes here
}
function myFunction2(xhttp) {
  // action goes here
}
```

3.5 Send a Request

To send a request to a server, you can use the open() and send() methods of the XMLHttpRequest object:

```
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

For security reasons, modern browsers do not allow access across domains. This means that both the web page and the XML file it tries to load, must be located on the same server.

3.5.1 GET or POST

GET is simpler and faster than POST, and can be used in most cases. However, always use POST requests when:

- A cached file is not an option (update a file or database on the server)
- Sending a large amount of data to the server (POST has no size limitations)
- Sending user input (which can contain unknown characters), POST is more robust and secure than GET

A simple GET request:

```
xhttp.open("GET", "demo_get.asp");
xhttp.send();
```

If you want to send information with the GET method, add the information to the URL:

```
xhttp.open("GET", "demo_get2.asp?fname=Henry&lname=Ford");
xhttp.send();
```

A simple POST request:

```
xhttp.open("POST", "demo_post.asp");
xhttp.send();
```

To POST data like an HTML form, add an HTTP header with setRequestHeader(). Specify the data you want to send in the send() method:

```
xhttp.open("POST", "ajax_test.asp");
xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xhttp.send("fname=Henry&lname=Ford");
```

3.5.2 The url - A File On a Server

The url parameter of the open() method, is an address to a file on a server. The file can be any kind of file, like .txt and .xml, or server scripting files like .asp and .php (which can perform actions on the server before sending the response back).

3.5.3 Asynchronous - True or False

Server requests should be sent asynchronously. The async parameter of the open() method should be set to true. By sending asynchronously, the JavaScript does not have to wait for the server response, but can instead:

- Execute other scripts while waiting for server response
- Deal with the response after the response is ready

The default value for the async parameter is async = true. You can safely remove the third parameter from your code. Synchronous XMLHttpRequest (async = false) is not recommended because the JavaScript will stop executing until the server response is ready. If the server is busy or slow, the application will hang or stop.

3.6 Server Response

3.6.1 Server Response Properties

Property	Description
responseText	get the response data as a string
responseXML	get the response data as XML data

The responseText property returns the server response as a JavaScript string, and you can use it accordingly:

```
document.getElementById("demo").innerHTML = xhttp.responseText;
```

The XMLHttpRequest object has an in-built XML parser. The responseXML property returns the server response as an XML DOM object. Using this property you can parse the response as an XML DOM object:

```
const xmlDoc = xhttp.responseXML;
const x = xmlDoc.getElementsByTagName("ARTIST");

let txt = "";
for (let i = 0; i < x.length; i++) {
  txt += x[i].childNodes[0].nodeValue + "<br>";
}
document.getElementById("demo").innerHTML = txt;

xhttp.open("GET", "cd_catalog.xml");
xhttp.send();
```

3.6.2 Server Response Methods

Property	Description
getResponseHeader()	Returns specific header information from the server resource
getAllResponseHeaders()	Returns all the header information from the server resource

The getAllResponseHeaders() method returns all header information from the server response.

```
const xhttp = new XMLHttpRequest();
xhttp.onload = function() {
    document.getElementById("demo").innerHTML =
    this.getAllResponseHeaders();
}
xhttp.open("GET", "ajax_info.txt");
xhttp.send();
```

The getResponseHeader() method returns specific header information from the server response.

```
const xhttp = new XMLHttpRequest();
xhttp.onload = function() {
    document.getElementById("demo").innerHTML =
    this.getResponseHeader("Last-Modified");
}
xhttp.open("GET", "ajax_info.txt");
xhttp.send();
```

4 AJAX XML Example

AJAX can be used for interactive communication with an XML file. The following example will demonstrate how a web page can fetch information from an XML file with AJAX:

```
<!DOCTYPE html>
<html>
<style>
table,th,td {
  border : 1px solid black;
  border-collapse: collapse;
}
th,td {
  padding: 5px;
}
</style>
<body>

<h2>The XMLHttpRequest Object</h2>

<button type="button" onclick="loadDoc()">Get my CD collection</button>
<br><br>
<table id="demo"></table>

<script>
function loadDoc() {
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {
    myFunction(this);
  }
  xhttp.open("GET", "cd_catalog.xml");
  xhttp.send();
}
function myFunction(xml) {
  const xmlDoc = xml.responseXML;
  const x = xmlDoc.getElementsByTagName("CD");
  let table="<tr><th>Artist</th><th>Title</th></tr>";
  for (let i = 0; i <x.length; i++) {
    table += "<tr><td>" +
    x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
```

```
    "</td><td>" +
    x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
    "</td></tr>";
  }
  document.getElementById("demo").innerHTML = table;
}
</script>

</body>
</html>
```

The output is:

Artist	Title
Bob Dylan	Empire Burlesque
Bonnie Tyler	Hide your heart
Dolly Parton	Greatest Hits

The XML file used in the example above looks like this:

```
<CATALOG>
<CD>
<TITLE>Empire Burlesque</TITLE>
<ARTIST>Bob Dylan</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Columbia</COMPANY>
<PRICE>10.90</PRICE>
<YEAR>1985</YEAR>
</CD>
<CD>
<TITLE>Hide your heart</TITLE>
<ARTIST>Bonnie Tyler</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>CBS Records</COMPANY>
<PRICE>9.90</PRICE>
<YEAR>1988</YEAR>
</CD>
<CD>
<TITLE>Greatest Hits</TITLE>
<ARTIST>Dolly Parton</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>RCA</COMPANY>
<PRICE>9.90</PRICE>
<YEAR>1982</YEAR>
</CD>
</CATALOG>
```

When a user clicks on the "Get CD info" button above, the loadDoc() function is executed. The loadDoc() function creates an XMLHttpRequest object, adds the function to be executed when the server response is ready, and sends the request off to the server. When the server response is ready, an HTML table is built, nodes (elements) are extracted from the XML file, and it finally updates the element "demo" with the HTML table filled with XML data.

5 Discussion & Conclusion

Based on the focused objective(s) to understand the concepts of AJAX, the additional lab exercise made me more confident in the fulfillment of the objectives(s).

6 Lab Task (Please implement yourself and show the output to the instructor)

Implement the AJAX XML Example problem on your own.

7 Lab Exercise (Submit as a report)

Implement the AJAX XML Example problem for responseText property.

8 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.