Unit 4
1.7 AJAX (Asynchronous JavaScript And XML)

1.7 AJAX (Asynchronous JavaScript And XML)

AJAX (Asynchronous JavaScript And XML) is based on JavaScript and HTTP requests. AJAX is all about making rich and user friendly web applications

AJAX is not different programming language it is rather a term used to describe an approach to designing and implementing web applications. In other words it is a new way to use existing standards. It is a way of creating better, faster and more interactive web applications (Some people call AJAX as JavaScript with steroids ☺).

1.7.1 Conventional Model vs AJAX Model

Conventional Model:

Users have become accustomed to blazing-fast responses in their desktop applications in most modern web pages. They get frustrated when a website can't offer the same immediate response. In conventional model For example , you fill some form, validate the inputs and submit the form. We wait for the server to respond and new page to load. This way application may run slowly and will have less use friendly experience because there could be a great deal of the lag between user interaction and application response.

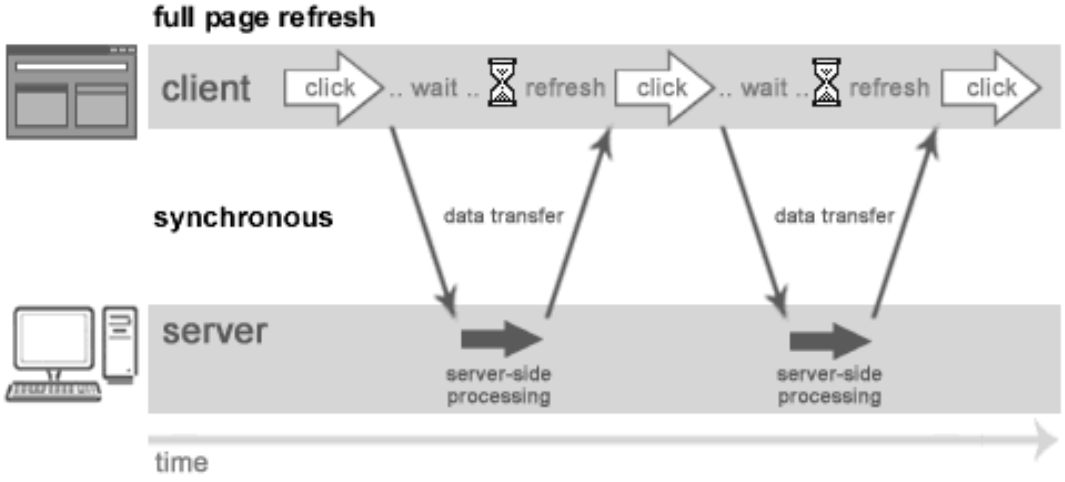

Figure 1.85 Classic Web Application Model:
Full page refresh and Synchronous Communication

AJAX Model:

By adding an additional layer between the user interface and the communication with the server, AJAX applications remove this lag between user interaction and application response. AJAX has the ability to directly communicate to the server and without page refresh, update the page by exchanging small amounts of data with the server behind the scenes. It uses asynchronous data transfer (we say it, HTTP requests) between the browser and web server. This means that it is possible to update parts of a web page, without reloading the whole page.

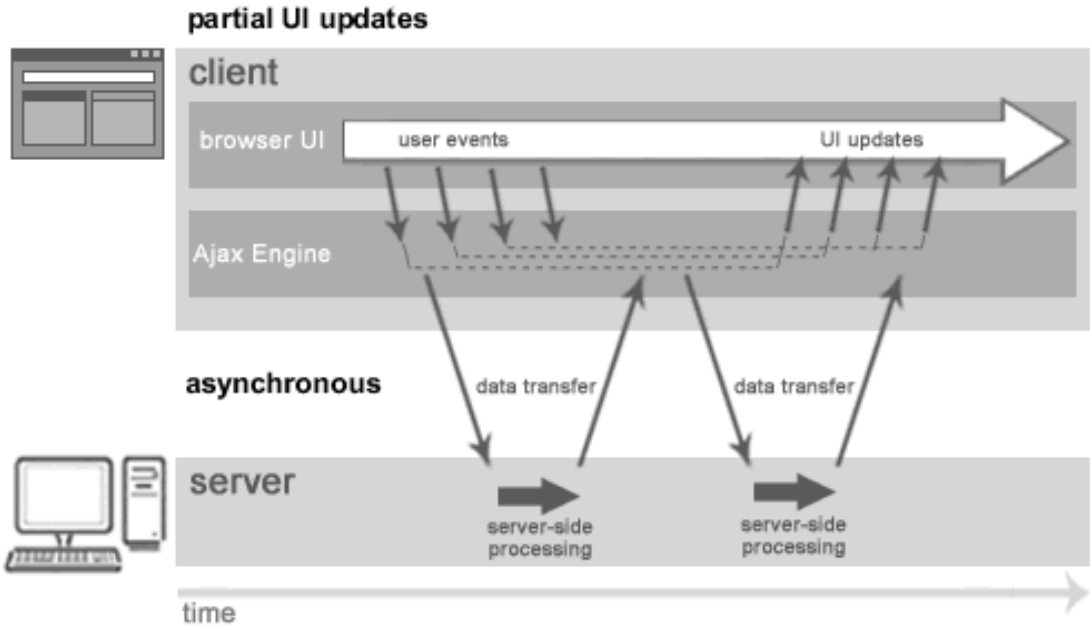

Figure 1.86 AJAX Model: Partial UI Updates and Asynchronous Communications

1.7.2 Google is heavily AJAX oriented

AJAX was made popular by Google with its Google suggestion box. When we start typing in Google's search box, a JavaScript sends the letters off to a server and the server returns a list of suggestions. This makes the user much easier to do the search he wants.

Some more web pages, using AJAX: Google Maps, Gmail, Youtube, and Facebook tabs.

1.7.3 AJAX supports Internet Standards.

AJAX is based on internet standards, and uses a combination of:

- XMLHttpRequest object (to exchange data asynchronously with a server)
- JavaScript/DOM (to display/interact with the information)
- CSS (to style the data)
- XML (often used as the format for transferring data)

AJAX – in most cases is part of client side code in JavaScript. Therefore AJAX code can 'mix' with HTML (XHTML) and Cascading Style Sheets (CSS) for building the underlying page structure and its visual style etc. So AJAX can create :

- Some sort of a interaction suite using the Document Object Model
- Data manipulation using Extensible Markup Language (XML)
- Data retrieval using XMLHttpRequest
- Use javaScript to help these different elements interact with one another.

1.7.4 AJAX Client Requests

To understand how AJAX works, we will create a small AJAX application.

Example 1.70

```
<html>
<head>
<script type="text/javascript">
var xmlDoc;
var xmlhttp;
function loadXMLDoc()
{
  xmlhttp = new XMLHttpRequest();
  xmlhttp.onreadystatechange = readData;
  xmlhttp.open("GET", "fileWithData.xml", true);
  xmlhttp.send();
}

function readData()
{
   if (xmlhttp.readyState == 4)
    {
       xmlDoc = xmlhttp.responseXML;
      var items = xmlDoc.getElementsByTagName(" aDataSet ");
      var nextValue1 = items[0].getAttribute("item1");
      var nextValue2  = items[0].getAttribute("item2");
      alert("Items Are : " +  nextValve1 + ", " + nextValue2);
   }
}
</script>
</head>
<body>

<div id="myDiv"><h2>Let AJAX change this text</h2></div>
<button type="button" onclick="loadXMLDoc()">Change Content</button>

</body>
</html>
```

This function must be called from html script

Register a callback function. Function defined below.

Provide the xml file that must be opened and read.

Call Back Function Defined

Extract data from the XML file and use it for your purposes.

XML file fileWithData.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<aDataSet>
  <dataline item1="some data 1" item2="some data 2"></dataline>
</ aDataSet >
```

In example 1.70, the AJAX application contains one div section and one button. The div section will be used to display information returned from a server. The button calls a function named loadXMLDoc(), if it is clicked.

1.7.4 AJAX Client Side Code for all major browsers

Previous example shows the general understanding of how AJAX works. However, difficult situations can encounter when AJAX agents execute in different browsers. The following skeleton has been tested in all three major browsers (IE, Firefox and Chrome) where a Microsoft.XMLDOM parameter is used to create a ActiveX object instead of a XMLHTTPRequest Object.

Example 1.71

```
<html>
<head>
<script type="text/javascript">
var xmlDoc;
var xmlhttp;
function loadXMLDoc()
{
  xmlhttp = new ActiveXObject("Microsoft.XMLDOM");
  xmlhttp.onreadystatechange = readData;
  xmlhttp.open("GET", "fileWithData.xml", true);
  xmlhttp.send();
}

function readData()
{
   if (xmlhttp.readyState == 4)
    {
       xmlDoc = xmlhttp.responseXML;
      var items = xmlDoc.getElementsByTagName(" aDataSet ");
      var nextValue1 = items[0].getAttribute("item1");
      var nextValue2  = items[0].getAttribute("item2");
      alert("Items Are : " +  nextValve1 + ", " + nextValue2);
   }
}
</script>
</head>
<body>

<div id="myDiv"><h2>Let AJAX change this text</h2></div>
<button type="button" onclick="loadXMLDoc()">Change Content</button>

</body>
</html>
```

Use a ActiveXObject with Microsoft.XMLDOM parameter instead of a XMLHTTPRequest object

The <script> tag to the page's head section contains the loadXMLDoc() function. In fact it is javascript.

- **The XMLHttpRequest Object**

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

- **Create an XMLHttpRequest Object**

All modern browsers (IE7+, Firefox, Chrome, Safari, and Opera) have a built-in **XMLHttpRequest** object.

Syntax for creating an XMLHttpRequest object:

xhttp=new XMLHttpRequest();

- **Sending Requests to the Server**

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

```
xhttp.open("GET"," “fileWithData.xml",true);
xhttp.send();
```

The following figure (Figure 1.87) shows the data flow path in AJAX.

Descriptions of the methods send() and open() are given below.

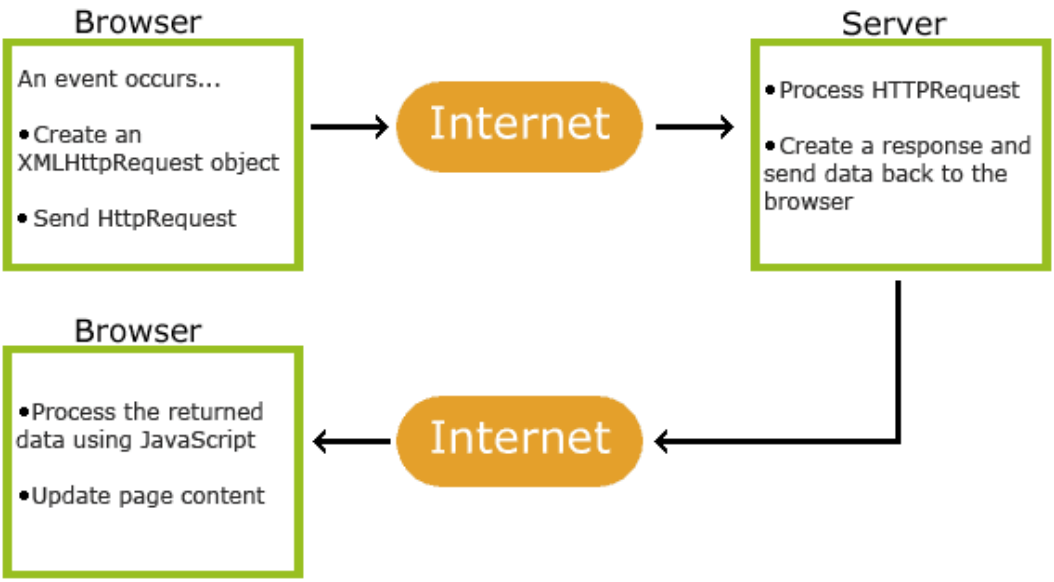

Figure 1.87 AJAX Data Flow Model

Method	Description
open(*method,url,async*)	Specifies the type of request, the URL, and if the request should be handled asynchronously or not.
	method: the type of request: GET or POST
send(*string*)	Sends the request off to the server.
	string: Only used for POST requests

Figure 1.88

- **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

- **GET Requests**

A simple GET request:

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

In the example above, you may get a cached result.
To avoid this, add a unique ID to the URL:

```
xhttp.open("GET","demo_get.asp?t=" + Math.random(),true);
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",true);
xhttp.send();
```

- **POST Requests**

A simple POST request:

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

Method	Description
setRequestHeader(*header,value*)	Adds HTTP headers to the request.
	header: specifies the header name
	value: specifies the header value

Figure 1.89

•The url - A File On a Server

The url parameter of the open() method, is an address to a file on a server:

xhttp.open("GET","ajax_test.asp",true);

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).

- **Asynchronous - True or False?**

AJAX stands for Asynchronous JavaScript and XML, and for the XMLHttpRequest object to behave as AJAX, the async parameter of the open() method has to be set to true:

```
xhttp.open("GET","ajax_test.asp",true);
```

Sending asynchronously requests is a huge improvement for web developers. Many of the tasks performed on the server are very time consuming. Before AJAX, this operation could cause the application to hang or stop.

With AJAX, 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 when the response ready

- **Async=true**

When using async=true, specify a function to execute when the response is ready in the onreadystatechange event:

```
xhttp.onreadystatechange=function()
  {
  if (xhttp.readyState==4 && xhttp.status==200)
    {
    document.getElementById("myDiv").innerHTML=xhttp.responseText;
    }
  }
xhttp.open("GET","ajax_info.txt",true);
xhttp.send();
```

- **Async=false**

To use async=false, change the third parameter in the open() method to false:

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

Using `async=false` is not recommended, but for a few small requests this can be ok.

Remember that the JavaScript will NOT continue to execute, until the server response is ready. If the server is busy or slow, the application will hang or stop.

Note: When you use `async=false`, do NOT write an `onreadystatechange` function - just put the code after the send() statement:

```
xhttp.open("GET","ajax_info.txt",false);
xhttp.send();
document.getElementById("myDiv").innerHTML=xhttp.responseText;
```

1.7.5 AJAX Server Response

To get the response from a server, use the `responseText` or `responseXML` property of the `XMLHttpRequest` object.

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

Figure 1.90

- **The responseText Property**

If the response from the server is not XML, use the `responseText` property. The `responseText` property returns the response as a string, and you can use it accordingly:

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

- **The responseXML Property**

If the response from the server is XML, and you want to parse it as an XML object, use the responseXML property:

Request the file cd_catalog.xml and parse the response:

```
xmlDoc=xhttp.responseXML;
var txt="";
x=xmlDoc.getElementsByTagName("ARTIST");
for (i=0;i<x.length;i++)
  {
  txt=txt + x[i].childNodes[0].nodeValue + "<br />";
  }
document.getElementById("myDiv").innerHTML=txt;
```

1.7.6 The `onreadystatechange` Event

When a request to a server is sent, we want to perform some actions based on the response. The `onreadystatechange` event is triggered every time the `readyState` changes. The `readyState` property holds the status of the `XMLHttpRequest`. Three important properties of the `XMLHttpRequest` object:

Property	Description
onreadystatechange	Stores a function (or the name of a function) to be called automatically each time the readyState property changes
readyState	Holds the status of the XMLHttpRequest. Changes from 0 to 4:
	0: request not initialized
	1: server connection established
	2: request received
	3: processing request
	4: request finished and response is ready
status	200: "OK"
	404: Page not found

Figure 1.91

In the `onreadystatechange` event, we specify what will happen when the server response is ready to be processed. When `readyState` is 4 and status is 200, the response is ready:

Example

```
xhttp.onreadystatechange=function() {
  if (xhttp.readyState==4 && xhttp.status==200) {
    document.getElementById("myDiv").innerHTML=xhttp.respon
seText;
    }
  }
```

Note: The `onreadystatechange` event is triggered four times, one time for each change in `readyState`.

- **Using a Callback Function**

A callback function is a function passed as a parameter to another function.

If you have more than one AJAX task on your website, you should create ONE standard function for creating the `XMLHttpRequest` object, and call this for each AJAX task.

The function call should contain the URL and what to do on `onreadystatechange` (which is probably different for each call):

```
Example
function myFunction()
{
loadXMLDoc("ajax_info.txt",function()
  {
  if (xhttp.readyState==4 && xhttp.status==200)
    {

    document.getElementById("myDiv").innerHTML=xhttp.respon
    seText;
     }
  });
}
```

AJAX is a very large lesson. Rest about AJAX is beyond the scope of this course. However, we might discuss little bit more about AJAX towards the end of the course.