1 What is Ajax?

1.1 Introduction

Asynchronous JavaScript and XML (Ajax) is a term for the process of transferring data between a client script and the server. The advantage of this is that it provides developers with a way to retrieve content from a Web server without reposting the page the user is currently viewing to the server. In concert with modern browsers’ ability to dynamically change displayed content through programming code (JavaScript) that accesses the browser's DOM, Ajax lets developers update the HTML content displayed in the browser without refreshing the page. Thus, Ajax provides dynamic interaction between a client and a server. In other words, Ajax can make browser-based applications more interactive, more responsive, and more like traditional desktop applications. Google's Gmail and Outlook Express are two familiar examples that use Ajax techniques. Ajax has various applications, some of which are discussed below.

1. Dynamic Form Data Validation. As an example, suppose a user fills out a form to register with a web site. The validity of data in the form is not checked till the form is submitted. With Ajax, the data added to the form is dynamically validated using business logic in a server application. Thus, a complete form does not have to be posted to the server to check if data in the form is valid.

2. Auto completion. As a user adds some data to a form, the remaining form gets auto completed.

3. Refreshing data on a page. Some web pages require that data be refreshed frequently, a weather web site for example. Using the Ajax technique, a web page may poll the server for latest data and refresh the web page without reloading the page.

Ajax is based on XMLHttpRequest, JavaScript and XML DOM technologies. JavaScript and XML DOM technologies are relatively old technologies. Therefore we won’t discuss these. XMLHttpRequest is a
relatively new technology. In the next section, we shall discuss the XMLHttpRequest technology.

1.2 What is XMLHttpRequest?

Ajax takes advantage of an object built into all modern browsers—the XMLHttpRequest object—to send and receive HTTP requests and responses. An HTTP request sent via the XMLHttpRequest object does not require the page to have or post a `<form>` element. The “A” in Ajax stands for “asynchronous”, which means that the XMLHttpRequest object’s `send()` method returns immediately, letting the browser processing of other HTML/JavaScript on the Web page continue while the server processes the HTTP request and sends the response. While asynchronous requests are the default, the reader can optionally send synchronous requests, which halt other Web page processing until the page receives the server's response.

Microsoft introduced the XMLHttpRequest object as an ActiveX object in Internet Explorer (IE) 5. Other browser manufacturers, recognizing the object's utility, implemented the XMLHttpRequest object in their browsers, but as a native JavaScript object rather than as an ActiveX object. In turn, recognizing the value and security in that implementation type, Microsoft has recast the XMLHttpRequest in IE 7 as a window object property. Even when the implementation (and thus invocation) details differ, all the browsers' implementations have similar functionality and essentially identical methods. The W3C is working to standardize the XMLHttpRequest object, releasing a working draft of the W3C specification.

This chapter discusses the XMLHttpRequest object API in detail, listing and explaining all the properties and methods.

1.3 XMLHttpRequest Object Properties

The XMLHttpRequest object exposes various properties, methods, and events so Ajax scripts can process and control HTTP requests and responses. The rest of this chapter discusses these in detail.

---

1 W3C XMLHttpRequest Specification- http://www.w3.org/TR/XMLHttpRequest/
1.3 XMLHttpRequest Object Properties

1.3.1 The readyState Property

The XMLHttpRequest object cycles through several states as it sends an HTTP request to the server, waits while the request is processed, and when it receives a response. So that scripts can respond appropriately to the various states, the object exposes a readyState property that represents the object's current state, as shown in Table 1.1.

<table>
<thead>
<tr>
<th>ReadyState Property Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Represents an “uninitialized” state in which an XMLHttpRequest object has been created, but not initialized.</td>
</tr>
<tr>
<td>1</td>
<td>Represents a “sent” state in which code has called the XMLHttpRequest open() method and the XMLHttpRequest is ready to send a request to the server.</td>
</tr>
<tr>
<td>2</td>
<td>Represents a “sent” state in which a request has been sent to the server with the send() method, but a response has not yet been received.</td>
</tr>
<tr>
<td>3</td>
<td>Represents a “receiving” state in which the HTTP response headers have been received, but message body has not yet been completely received.</td>
</tr>
<tr>
<td>4</td>
<td>Represents a “loaded” state in which the response has been completely received.</td>
</tr>
</tbody>
</table>

1.3.2 The onreadystatechange Property

The XMLHttpRequest object generates a readystatechange event whenever the readyState value changes. The onreadystatechange property accepts an EventListener value, specifying the method that the object will invoke whenever the readyState value changes.

1.3.3 The responseText Property

The responseText property contains the text of the HTTP response received by the client. When the readyState value is 0, 1, or 2 responseText contains an empty string. When the readyState value is 3 (Receiving), the response contains the incomplete response
received by the client. When `readyState` is 4 (Loaded) the `.responseText` contains the complete response.

### 1.3.4 The `responseXML` Property

The `responseXML` property represents the XML response when the complete HTTP response has been received (when `readyState` is 4), when the Content-Type header specifies the MIME (media) type as `text/xml`, `application/xml`, or ends in `+xml`. If the Content-Type header does not contain one of these media types, the `responseXML` value is `null`. The `responseXML` value is also `null` whenever the `readyState` value contains any value other than 4. The `responseXML` property value is an object of type `Document` interface, and represents the parsed document. If the document cannot be parsed (for example if the document is malformed or the character encoding of the document is not supported) the `responseXML` value is `null`.

### 1.3.5 The `status` Property

The `status` property represents the HTTP status code\(^2\) and is of type `short`. The `status` attribute is available only when the `readyState` value is 3 (Receiving) or 4 (Loaded). Attempting to access the `status` value when `readyState` is less than 3 raises an exception.

### 1.3.6 The `statusText` Property

The `statusText` attribute represents the HTTP status code text and is also available only when the `readyState` value is 3 or 4. Attempting to access the `statusText` property for other `readyState` values raises an exception.

---

\(^2\) Status Code Definitions- [http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html](http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html)
1.4 XMLHttpRequest Object Methods

The XMLHttpRequest object provides various methods to initiate and process HTTP requests, which are discussed in detail in the following sections.

1.4.1 The abort() Method

The abort() method is used to halt the HTTP request associated with an XMLHttpRequest object to reset the object to the uninitialized state.

1.4.2 The open() Method

The open(DOMString method, DOMString uri, boolean async, DOMString username, DOMString password) method is called to initialize an XMLHttpRequest object. The method parameter is required and specifies the HTTP method (GET, POST, PUT, DELETE, or HEAD) that want to use to send the request. To send data to the server, use the POST method. To retrieve data from the server, use the GET method. The uri parameter specifies the server URI to which the XMLHttpRequest object sends the request. The uri resolves to an absolute URI using the window.document.baseURI property—in other words, relative URIs will be resolved in the same way that the browser resolves relative URIs. The async parameter specifies whether the request is asynchronous; the default value is true. To send a synchronous request, set the parameter to false. For servers that require authentication, the optional username and password parameters may be supplied. After calling the open() method, the XMLHttpRequest object sets its readyState property to 1 (Open) and resets the responseText, responseXML, status, and statusText properties to their initial values. It also resets the request headers. Note that the object resets these values if the open() method when readyState is 4.

1.4.3 The send() Method

After preparing a request by calling the open() method, the request is sent to the server. The send() method may be called only when the readyState value is 1, otherwise the XMLHttpRequest object raises an exception. The request gets sent to the server using the parameters supplied to the open() method. The send() method returns immediately when the async parameter is true, letting other client
script processing continue. The XMLHttpRequest object sets the readyState value to 2 (Sent) after the send() method has been called. When the server responds, before receiving the message body, if any, the XMLHttpRequest object sets readyState to 3 (Receiving). When the request has completed loading it sets readyState to 4 (Loaded). For a request of type HEAD, it sets the readyState value to 4 immediately after setting it to 3.

The send() method takes an optional parameter that may contain data of varying types. Typically, this method is used to send data to the server using the POST method. The send() method may be explicitly invoked with null, which is the same as invoking it with no argument. For most other data types, set the Content-Type header using the setRequestHeader() method (explained below) before invoking the send() method. If the data parameter in the send(data) method is of type DOMString, encode the data as UTF-8. If data is of type Document, serialize the data using the encoding specified by data.xmlEncoding, if supported or UTF-8 otherwise.

### 1.4.4 The setRequestHeader() Method

The setRequestHeader(DOMString header, DOMString value) method sets request headers. This method may be called after calling the open() method-when the readyState value is 1-otherwise you'll get an exception.

### 1.4.5 The getResponseHeader() Method

The getResponseHeader(DOMString header, value) method is used to retrieve response header values. Call getResponseHeader() only when the readyState value is 3 or 4 (in other words, after the response headers are available); otherwise, the method returns an empty string.

### 1.4.6 The getAllResponseHeaders() Method

The getAllResponseHeaders() method returns all the response headers as a single string with each header on a separate line. The method returns null if readyState value is not 3 or 4.
1.5 Sending an Ajax Request

In Ajax, many requests that use the XMLHttpRequest are initiated from a HTML Event such as a button click (onclick) or a key press (onkeypress) that invokes a JavaScript function. Ajax has various applications including form validation. Sometimes a unique form field is required to be validated before the rest of the form may be filled. For example a registration form that requires a unique UserID. Without validation of the UserID field with Ajax the complete form would have to be filled and submitted. If the UserID is not valid, the form would have to be re-submitted. For example, a form field for a Catalog ID that must be validated on the server might be specified as follows.

```html
<form name="validationForm" action="validateForm" method="post">
  <table>
    <tr><td>Catalog Id:</td><td><input type="text" size="20" id="catalogId" name="catalogId" onkeyup="sendRequest()"></td><td><div id="validationMessage"></div></td></tr>
  </table></form>
```

The preceding HTML uses the validationMessage div to display a validation message for the input field Catalog Id. The onkeyup event invokes a JavaScript sendRequest() function. The sendRequest() function creates an XMLHttpRequest object. The process of creating an XMLHttpRequest object differs depending on the browser implementation. If the browser supports the XMLHttpRequest object as a window property (all common browsers do except IE 5 and 6), the code can call the XMLHttpRequest constructor. If the browser implements the XMLHttpRequest object as an ActiveXObject object (as in IE versions 5 and 6), the code uses the ActiveXObject constructor. The function below calls an init() function, which checks to determine the appropriate creation method to use before creating and returning the object.

```javascript
function sendRequest(){
  var xmlHttpReq=init();
}
function init() {
  if (window.XMLHttpRequest) {
    return new XMLHttpRequest();
  }
  else if (window.ActiveXObject) {
    return new ActiveXObject("Microsoft.XMLHTTP");
  }
}
</script>

Next, we need to initialize the XMLHttpRequest object using the open() method, specifying the HTTP method and the server URL to use.

```javascript
var catalogId=encodeURIComponent(document.getElementById("catalogId").value);
xmlHttpReq.open("GET", "validateForm?catalogId=" + catalogId, true);
```

HTTP requests sent with XMLHttpRequest are asynchronous by default, but the async parameter may be explicitly set to true as shown above.

In this case, the call to the URL validateForm invokes a servlet on the server-side, but it should recognize that the server-side technology is immaterial; the URL might actually be an ASP, ASP.NET, or PHP page, or a Web service—it doesn’t matter as long as the page returns a response indicating whether the CatalogID value is valid. Because you're making an asynchronous call, we need to register a callback event handler that the XMLHttpRequest object will invoke when its.readyState value changes. Remember that a change to the readyState value generates a readystatechange event. We register the callback event handler using the onreadystatechange property.

```javascript
xmlHttpReq.onreadystatechange=processRequest;
```

Next, we need to send the request using the send() method. Because this request uses the HTTP GET method, the send() method may be invoked without an argument or null argument.

```javascript
xmlHttpReq.send(null);
```

### 1.6 Processing an Ajax Request

In this example, because the HTTP method is GET, the receiving servlet on the server invokes a doGet() method, which retrieves the catalogId parameter value specified in the URL, and checks its validity against a database. The servlet needs to construct a response to be sent to the client.
This example returns XML, so it sets the HTTP content type of the response to text/xml and the Cache-Control header to no-cache. Setting the cache-control header prevents the browser from simply reloading the page from the cache.

```java
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    ...
    response.setContentType("text/xml");
    response.setHeader("Cache-Control", "no-cache");
}
```

The response from the server is an XML DOM object. Create an XML string that contains instructions to be processed on the client side. The XML string must have a root element.

```java
out.println("<catalogId>valid</catalogId>");
```

The XMLHttpRequest object is designed to handle responses consisting of plain text or XML; but a response may be of another type if the user agent (UA) supports the content type.

The XMLHttpRequest object calls the event handler registered with onreadystatechange when the request state changes. Therefore, your event handler should check the readyState value and the HTTP status before processing the response. When the request has completed loading, which corresponds to readyState value 4, and the HTTP status is “OK”, the response has completed, and we may invoke a JavaScript function to process the response content. The following script checks the values and invokes a processResponse() method when the response is complete.

```javascript
function processRequest(){
    if(xmlHttpReq.readyState==4){
        if(xmlHttpReq.status==200){
            processResponse();
        }
    }
}
```

The processResponse() method uses the XMLHttpRequest objects' responseXML and responseText properties to retrieve the HTTP response. As explained above, the responseXML is available only if the media type of the response is text/xml, application/xml or ends in +xml. The responseText property
returns the response as plain text. For an XML response we would retrieve the content as follows.

```javascript
var msg=xmlHttpReq.responseXML;

With the XML stored in the `msg` variable, we retrieve the element's value using the DOM method `getElementsByTagName()`.

```javascript
var catalogId=msg.getElementsByTagName("catalogId")[0].firstChild.nodeValue;
``` 

Finally, we test the element value to create a message that we display by updating the HTML content of the `validationMessage` div on the Web page, using the `innerHTML` property.

```javascript
if(catalogId=="valid"){
    var validationMessage = document.getElementById("validationMessage");
    validationMessage.innerHTML = "Catalog Id is Valid";
} else {
    var validationMessage = document.getElementById("validationMessage");
    validationMessage.innerHTML = "Catalog Id is not Valid";
}
```

That's the full cycle. The `XMLHttpRequest` object provides dynamic interaction between a client and a server by transferring data without posting the Web page to the server. We use JavaScript to launch a request and process the return value, and then we use browser DOM methods to update data on the page. We are using Oracle JDeveloper 11g IDE for Ajax, because JDeveloper 11g provides an integrated JavaScript Editor for Ajax/web development. We shall discuss the JavaScript Editor next.

### 1.7 JDeveloper Integrated JavaScript Editor

JDeveloper 11g includes an integrated JavaScript editor for creating JavaScript. In a JDeveloper web application JavaScript may be added directly to a JSP file, but the JavaScript may also be created in a separate `.js` file and the `.js` file added to the JSP using the `<script/>` tag.
Creating the JavaScript file separately has an advantage as the integrated JavaScript Editor may be availed of. Create a JavaScript file by selecting File>New and in the New Gallery window Web Tier>HTML>JavaScript File. Copy some JavaScript code to the JavaScript file. Create a JSP file to add the JavaScript file to with File>New. In the New Gallery window select Web Tier>JSP in Categories and select JSP in Items. The JavaScript file and the JSP file are shown in Fig. 1.1.

One of the features of the JavaScript editor is syntax highlighting. To add syntax highlighting select Tools>Preferences and in the Preferences window select Code Editor>Syntax Colors. Select JavaScript in the Language list. The Font Style, Foreground color and Background color may be set for the different JavaScript constructs in the Syntax Colors window as shown in Fig. 1.2.
1 What is Ajax?

Fig. 1.2 Setting Syntax Highlighting

JavaScript editor also provides code completion. As the JavaScript syntax varies in the different browsers we need to specify the browser for which code completion is to be implemented. Select JavaScript Editor in the Preferences window and select a Target Browser for code completion as shown in Fig. 1.3.

Fig. 1.3 Selecting Target Browser
In the JavaScript file right-click and select **Source>Completion Insight** or **Source>Smart Completion Insight** for code insight as shown in Fig. 1.4.

**Fig. 1.4 Using Code Insight**

Another feature of the JavaScript editor is **Go To Declaration** using which a JavaScript variable or function may be navigated to from a usage of the JavaScript variable/function. For example, select a usage of the variable `xmlHttpRequest`, right-click and select **Go To Declaration** to go to the declaration of the `xmlHttpRequest` variable as shown in Fig. 1.5.
JavaScript editor also provide brace matching and code folding. Another feature is error underling and error auditing. For example, add an error by removing the ‘{‘ for a function declaration. An error analysis gets run and the errors get underlined as shown in Fig. 1.6.
Usages of a variable or function may be retrieved by selecting the variable/function and selecting **Find Usages**. For example, select `xmlHttpRequest`, right-click and select **Find Usages**. The usages of the `xmlHttpRequest` variable get listed in the log window as shown in Fig. 1.7.

![Fig. 1.7 Find Usages](image)

A JavaScript file is integrated with the **Structure** pane from which different variables and functions may be navigated to as shown in Fig. 1.8.
1 What is Ajax?

Fig. 1.8 JavaScript File Structure

JavaScript editor also provides refactoring to rename or delete a variable or function. To refactor, right-click and select **Refactor> Rename** or **Refactor> Delete Safely** as shown in Fig. 1.9.

Fig. 1.9 Refactoring
To add the JavaScript file to a JSP drag and drop the file from the Application navigator to the JSP. A `<script/>` element for the JavaScript file gets added to the JSP as shown in Fig. 1.10.

![Fig. 1.10 Adding JavaScript to JSP](image)

### 1.8 Summary

In this chapter we discussed the XMLHttpRequest object, which forms the basis of Ajax. An Ajax request is initiated from a browser by first creating an XMLHttpRequest object and opening the XMLHttpRequest object using the `open()` method. The method used to create the XMLHttpRequest varies with the browser used. An Ajax request is sent using the `send()` method of XMLHttpRequest. When the request completes the Ajax XML response is retrieved using the `responseXML` attribute of the XMLHttpRequest object. The web page that sent the Ajax request is updated with the Ajax XML response by retrieving the XML data and setting the data into the web page elements using DOM functions. We also discussed the JavaScript Editor integrated into JDeveloper 11g.
Ajax in Oracle JDeveloper
Vohra, D.
2008, XIV, 224 p., Softcover
ISBN: 978-3-540-77595-9