Build apps using Asynchronous JavaScript with XML (AJAX)

Learn to construct real time validation-enabled Web applications with AJAX

Skill Level: Introductory

Naveen Balani (naveenbalani@rediffmail.com)
Technical Architect
Webify Solutions

Rajeev Hathi (rajeev_hathi@hotmail.com)
Senior Systems Analyst
Satyam Computers Ltd.

15 Nov 2005

AJAX (Asynchronous JavaScript with XML) enables a dynamic, asynchronous Web experience without the need for page refreshes. In this tutorial, you learn to build AJAX-based Web applications -- complete with real time validation and without page refreshes -- by following the construction of a sample book order application.

Section 1. Before you start

About this tutorial

In this tutorial, we explain how to develop and design Web applications based on Asynchronous JavaScript with XML, or AJAX. You'll build a sample Web-based *book order application* which provides real time validation and page refresh, for efficient and smooth user interaction.

Prerequisites

We will use Tomcat to run the AJAX application. Tomcat is the servlet container that

is used in the official reference implementation for the Java Servlet and JavaServer Pages technologies. Download jakarta-tomcat-5.0.28.exe from the Jakarta Site and run it to install Tomcat to any location you'd like -- c:\tomcat5.0, for instance.

Download the source code and Web application (in wa-ajax-Library.war) for this tutorial.

Section 2. Introduction to AJAX

AJAX basics

AJAX enables a dynamic, asynchronous Web experience without the need for page refreshes. It incorporates the following technologies:

- XHTML and CSS provide a standards-based presentation.
- Document Object Model (DOM) provides dynamic display and interaction.
- XML and XSLT provide data interchange and manipulation.
- `XMLHttpRequest` provides asynchronous data retrieval.
- JavaScript binds everything together.

The core of AJAX technology is a JavaScript object: `XMLHttpRequest`. This object is supplied through browser implementations -- first through Internet Explorer 5.0 and then through Mozilla-compatible browsers. Take a closer at this object.

XMLHttpRequest

With `XMLHttpRequest`, you can use JavaScript to make a request to the server and process the response without blocking the user. As you create your Web site and use `XMLHttpRequest` to perform screen updates on the client's browser without the need for refresh, it provides much flexibility and a rich user experience.

Examples of `XMLHttpRequest` applications include Google's Gmail service, Google's Suggest dynamic lookup interface, and the MapQuest dynamic map interface. In the next sections, we demonstrate how to use `XMLHttpRequest` object in detail as we demonstrate the design of a book order application and implement it.

Section 3. Application design

Elements of the application

The sample Web-based book order application will contain the following client-side functions implemented in AJAX:

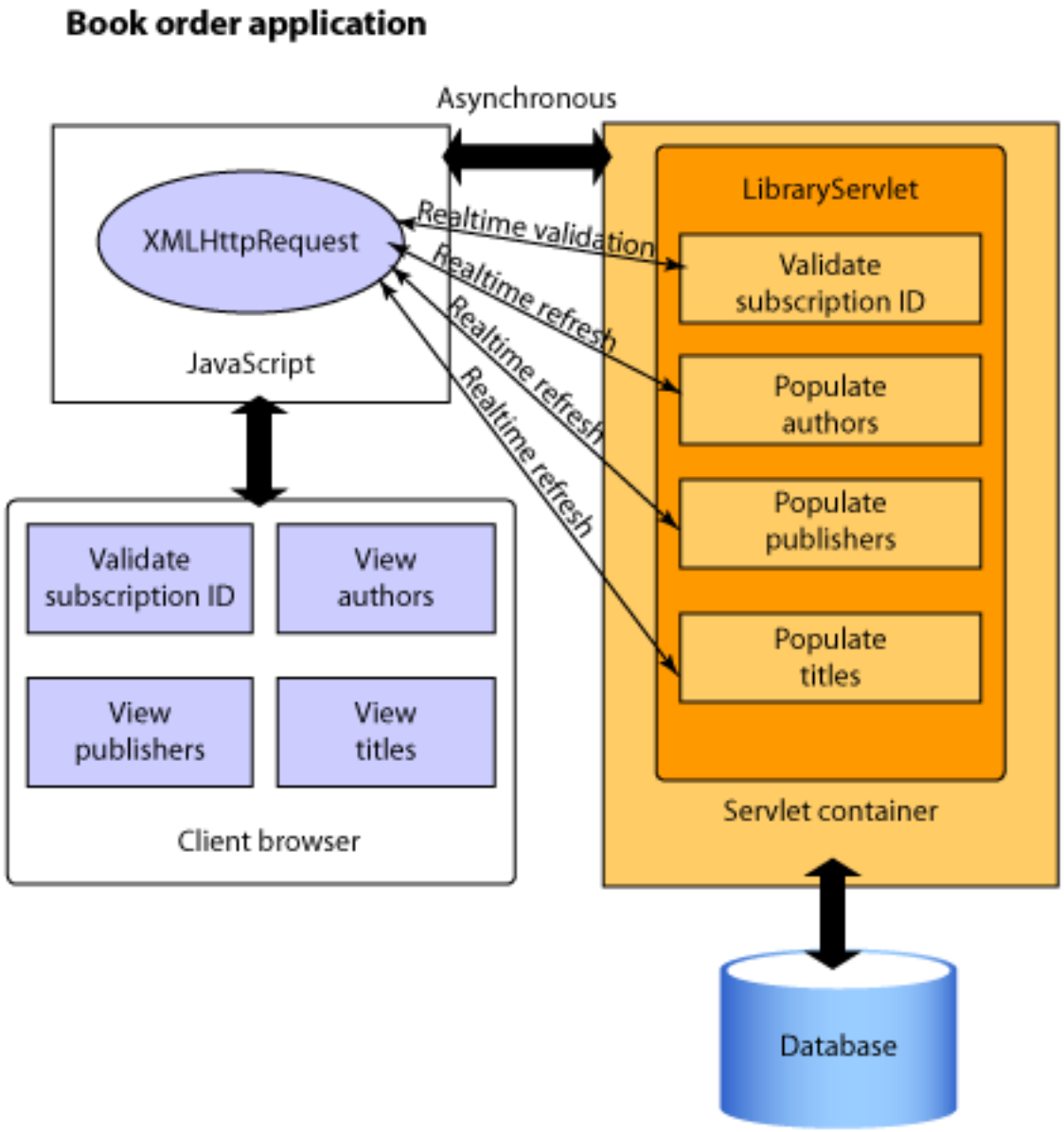
- Subscription ID validation
- A *View Authors* list
- A *View Publishers* list

The objective here is to show how *real time validation* and *page refreshes* in a Web page make user interaction smoother and more efficient.

Structure of the application

The diagram in Figure 1 depicts the design architecture of the sample book order application:

Figure 1. The AJAX architecture



The application will be a single Web page developed with JavaServer Pages (JSP) technology. The user will be able to invoke the Web page using a Web browser (such as Microsoft® Internet Explorer) and enter the *Subscription ID* which the application validates in real time. As the ID is validated asynchronously, the user can input more information. The user can view the book titles either by *Author* or *Publisher*. The screen will populate the *Authors* or *Publishers list* based on user choice. Based on the selection, the *Titles list* is populated. All these lists will populate in real time -- in other words, the page is not refreshed, but still the data comes from the backend tier. We call this phenomenon *real time refreshes*.

As you can see in Figure 1, the `XMLHttpRequest` JavaScript object helps with the real time asynchronous processing. This object makes a request in the form of XML over HTTP to the `LibraryServlet` servlet residing in a Web container. The servlet then queries the database, fetches the data, and sends it back to the client, again in the form of XML over HTTP. The requests and responses occur in real time without

refreshing the page.

This is what makes AJAX so powerful. The user does not wait for page reload to complete because there is no page reload.

In the next section, we'll demonstrate how to implement the book order application based on this design. We will take you through the code and perform some analysis. (To get the sample code for this tutorial, download the file, x-ajax-library.war.)

Section 4. Implementing the application

Application implementation with AJAX

In this section, we do a code walkthrough of the sample book order application and take a close look at each AJAX-based, Javascript component:

- Validate Subscription ID
- View Authors
- View Publishers
- View Titles

Code walkthrough: Validate the subscription ID

Let's start with the function *Validate Subscription ID* `<input type="text" name="subscriptionID" onblur="validate(this.form)"/>`. This code creates a text field where users can enter *Subscription ID*s. Once the user enters the ID and moves to the next field in the form, the `onBlur` event fires. This event calls a JavaScript function `validate()`:

```
var req;
function validate(formObj) {
    init();
    req.onreadystatechange = subscriptionValidator;
    req.send("subscriptionID=" + formObj.subscriptionID.value);
}
```

The `validate()` function takes `formObj` as a parameter. It first calls the `init()` function:

```
function init() {
    if (window.XMLHttpRequest) {
        req = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        req = new ActiveXObject("Microsoft.XMLHTTP");
    }
    var url = "/Library/LibraryServlet";
    req.open("POST", url, true);
```

```
    req.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
}
```

Code walkthrough: init()

Now look at the `init()` function does (we divide the code in parts):

```
if (window.XMLHttpRequest) {
    req = new XMLHttpRequest();
} else if (window.ActiveXObject) {
    req = new ActiveXObject("Microsoft.XMLHTTP");
}
```

The `init()` function first creates the `XMLHttpRequest` object. This request object is the core of AJAX. It sends and receives the request in XML form. This piece of code checks for browser support for the `XMLHttpRequest` object (most browsers support it). If you are using Microsoft Internet Explorer 5.0 or above, then the second condition is executed.

```
req.open("POST", url, true);
req.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
```

Once your code creates the `XMLHttpRequest` object, you need to set certain request properties. In the preceding code, the first line sets the request method, request URL, and the type of request (whether it is asynchronous or not). It does so by calling the `open()` method on the `XMLHttpRequest` object.

Here we will use the `POST` method. Ideally, use `POST` when you need to change the state on the server. Our application is not going to change the state, but we still prefer to use `POST`. The `url` is the URL of the servlet to be executed. `true` indicates that we will process the request asynchronously.

For the `POST` method, we need to set the request header `Content-Type`. This is not required for the `GET` method.

```
function validate(formObj) {
    init();
    req.onreadystatechange = subscriptionValidator;
    req.send("subscriptionID=" + formObj.subscriptionID.value);
}
```

Code walkthrough: Callback handler 1

To continue with the validation method, next you assign the `subscriptionValidator` callback handler to `onreadystatechange` which will fire at every state change on the request.

What is this *callback handler* all about? Since you are processing the request asynchronously, you need a callback handler which is invoked when the complete response is returned from the server -- the callback handler is where you will validate the subscription ID (that is, write your actual validation code).

The handler acts as a listener. It waits until the response is complete. (More on the handler code later.) To send the request, the last line calls the `send()` method. The

request is sent as a *name=value* pair. For the `GET` method, the request is sent as part of the URL, so the `send()` method is passed a null parameter.

The request is sent to the servlet. The servlet processes the request and sends back the response in real time. This is how the servlet processes the request. The next code snippet illustrates the `LibraryServlet -- doPost()` method.

```
public void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
    String ID = null;
    ID = req.getParameter("subscriptionID");
    if (ID != null) {
        String status = "<message>" + this.validID(ID) + "</message>";
        this.writeResponse(resp, status);
    }
}
```

Code walkthrough: Callback handler 2

The `doPost()` method gets the `subscriptionID` from the request parameter. To validate the ID, it calls the `validID()` method. This method validates the ID and returns `true` if the ID is valid, otherwise it returns `false`. It constructs the return status in XML format and writes the response by calling the `writeResponse()` method. Now examine the `writeResponse()` method.

```
public void writeResponse(HttpServletResponse resp, String output) throws IOException {
    resp.setContentType("text/xml");
    resp.setHeader("Cache-Control", "no-cache");
    resp.getWriter().write(output);
}
```

The response is sent in XML format. The first line sets the response content type, which is `text/xml`. The next line sets the header `Cache-Control` with the value of `no-cache`. This header is mandatory. AJAX requires that response output is not cached by the browser. To write the response, the last line calls the `getWriter().write()` method.

Code walkthrough: Callback handler 3

The request is processed by the servlet and the response is sent back to the client. Remember, this all happens in the background without a page refresh. Now the callback handler method that we discussed earlier, will handle and parse the response:

```
function subscriptionValidator() {
    if (req.readystate == 4) {
        if (req.status == 200) {
            var messageObj = req.responseXML.getElementsByTagName("message")[0];
            var message = messageObj.childNodes[0].nodeValue;
            if (message == "true") {
                msg.innerHTML = "Subscription is valid";
                document.forms[0].order.disabled = false;
            } else {
                msg.innerHTML = "Subscription not valid";
                document.forms[0].order.disabled = true;
            }
        }
    }
```

```
}
```

Code walkthrough: Revisiting XMLHttpRequest

As mentioned earlier, the `XMLHttpRequest` object is the core object which constructs and sends the request. It also reads and parses the response coming back from the server. Look at the code in parts.

```
if (req.readystate == 4) {
    if (req.status == 200) {
```

The preceding code checks the state of the request. If the request is in a ready state, it will then read and parse the response.

What do we mean by ready state? When the request object attribute `readystate` has the value of `4`, it means that the client received the response and is complete. Next we check the request status (whether the response was a normal page or an error page). To ensure that the response is normal, check for the status value of `200`. If the `status` value is `200`, then it will process the response.

```
var messageObj = req.responseXML.getElementsByTagName("message")[0];
var message = messageObj.childNodes[0].nodeValue;
if (message == "true") {
    msg.innerHTML = "Subscription is valid";
    document.forms[0].order.disabled = false;
} else {
    msg.innerHTML = "Subscription not valid";
    document.forms[0].order.disabled = true;
} }
```

Next, the request object reads the response by calling `responseXML` property. Note the servlet sent back the response in XML so we use `responseXML`. If the response sent was in text, then you can use the `responseText` property.

In this example, we deal with XML. The servlet constructed the response in a `<message>` tag. To parse this XML tag, call the `getElementsByTagName()` method on the `responseXML` property of the `XMLHttpRequest` object. It gets the tag name and the child value of the tag. Based on the value parsed, the response is formatted and written in HTML.

You just finished validating the subscription ID, all without a page refresh.

Code walkthrough: View authors, publishers, titles

The other functionalities -- *View Authors*, *View Publishers*, and *View Titles* -- work along similar lines. You have to define separate handlers for each functionality:

```
function displayList(field) {
    init();
    titles.innerHTML = " ";
    req.onreadystatechange = listHandler;
    req.send("select=" + escape(field));
}

function displayTitles(formObj) {
    init();
```

```
    var index = formObj.list.selectedIndex;
    var val = formObj.list.options[index].value;
    req.onreadystatechange = titlesHandler;
    req.send("list=" + val);
}
```

Remember, this sample application allows the user to view the titles by author or publisher. So either the *Author list* or the *Publisher list* displays. In such a scenario, the application only calls one callback handler based on the user selection -- in other words, for author and publisher list, you have only one `listHandler` callback handler.

To display the titles list, you will use `titlesHandler`. The remaining functionality stays the same with the servlet processing the request and writing back the response in XML fornat. The response is then read, parsed, formatted, and written in HTML. You can render the list in HTML as a `select.....options` tag. This sample code snippet shows the `titlesHandler` method.

```
var temp = "<select name=\"titles\" multiple\>";
for (var i=0; i<index; i++) {
    var listObj = req.responseXML.getElementsByTagName("list")[i];
    temp = temp + "<option value=" + i +">" + listObj.childNodes[0].nodeValue
+ "</option>";
}
temp = temp + "</select>";
titles.innerHTML = temp;
```

So far, we've demonstrated how to implement real time validation and refreshes. WIth AJAX, you can choose among several ways to add spice and flair to user interactions on your Web sites. Next we'll run the application.

Section 5. Running and testing the application

Run the application

Download the sample code wa-ajax-Library.war and copy it to your Tomcat Webapp directory (for example, c:\Tomcat 5.0\Webapps). To start the Tomcat server, type the following:

```
cd bin
C:\Tomcat 5.0\bin> catalina.bat start
```

Tomcat is now started with your AJAX Web application deployed.

Test the application

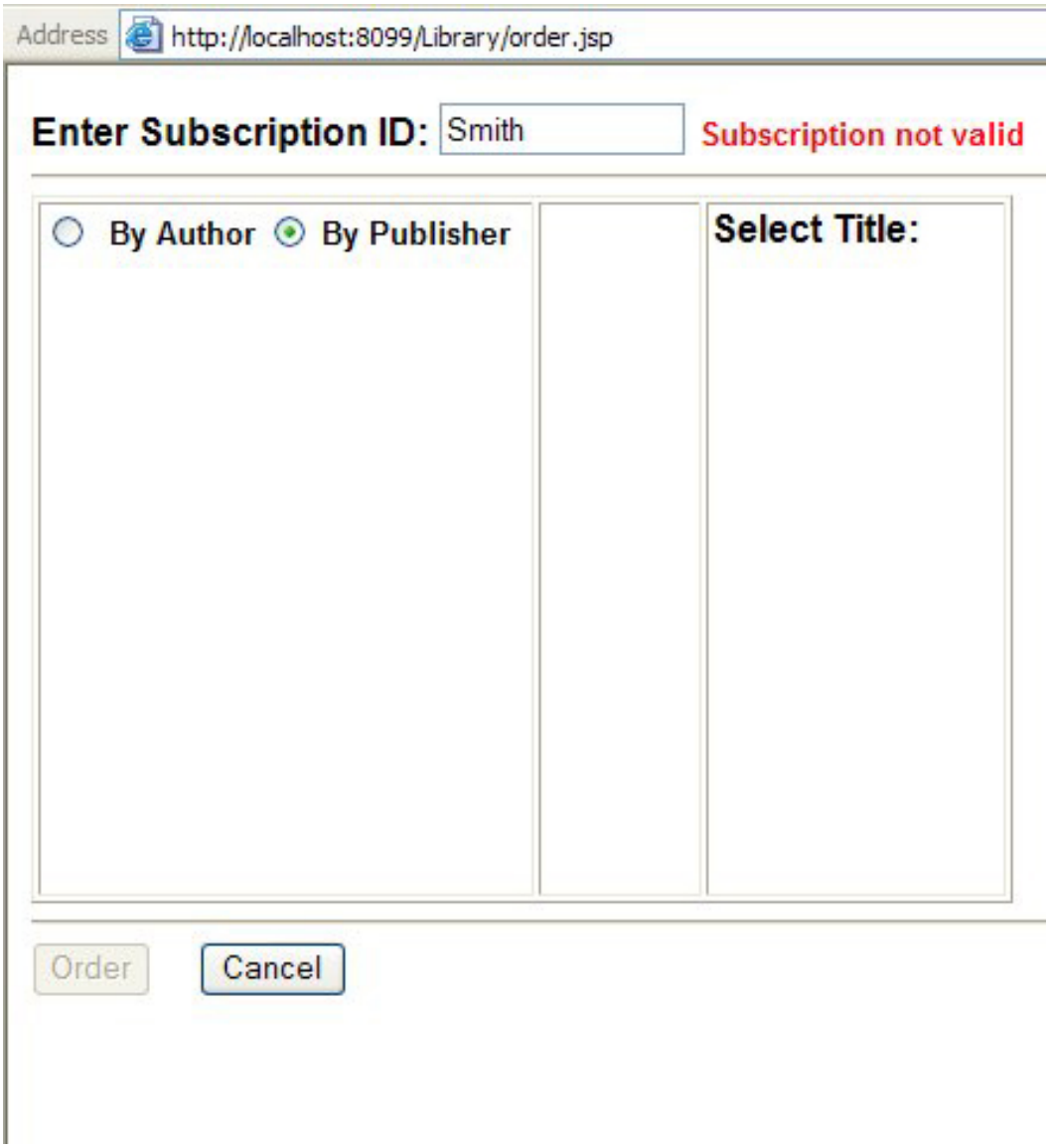
To test the application:

1. Open your Web browser. Point to

`http://localhost:tomcatport/Library/order.jsp` where the variable *tompcatport* is the port that your Tomcat server runs on.
You will see the subscription screen.

2. In the **Enter Subscription ID** field, type any user ID except "John" and tab out of the field.
The subscription ID request that you made to the server asynchronously will be validated. You will see a message "Subscription not valid" as shown in Figure 2:

Figure 2. The "Subcription not valid" screen



The application validated the user asynchronously and provided runtime validation without refreshing the browser.

3. Type in the user ID value, **John**.
You will see a message "Subscription is valid". Once the subscription is

valid, the application enables **Order** button.

4. Select the **By Author** or **By Publisher** radio button to populate the author or publisher drop-down list, respectively.

5. Select an author or publisher from the drop-down list.
The title area is populated dynamically (as in Figure 3).

Figure 3. The "Subcription is valid" screen

When you select the author or publisher, the application requests the server to provide the title information associated with the selected author or publisher at runtime from the server. The title information displays without refreshing the browser.

You've successfully installed and tested this sample AJAX Application.

Section 6. Summary

In conclusion

AJAX has come a long way since its inception. We believe AJAX can be applied as more than just a design pattern, though AJAX still has some issues:

- Browser support for the `XMLHttpRequest` object can be constraining. Most browsers do support the `XMLHttpRequest` object, but a few do not (usually the older version of browsers).
- AJAX is best suited for displaying a small set of data. If you deal with large volumes of data for a real time display of lists, then AJAX might not be the right solution.
- AJAX is quite dependent on JavaScript. If a browser doesn't support JavaScript or if a user disables the scripting option, then you cannot leverage AJAX at all.
- The asynchronous nature of AJAX will not guarantee synchronous request processing for multiple requests. If you need to prioritize your validation or refreshes, then design your application accordingly.

Even with these potential hiccups, AJAX still stands as the best solution to enhance your Web pages and resolve page-reload issues.

Resources

Learn

- Ajax for Java developers: Build dynamic Java applications (developerWorks, September 2005): This article introduces a groundbreaking approach to creating dynamic Web application experiences that solve the page-reload dilemma.
- XML Matters: Beyond the DOM (developerWorks, May 2005): Get details on the Document Object Model (DOM) as a method to build dynamic Web applications.
- Using Ajax with PHP and Sajax (developerWorks, October 2005): Take this tutorial if you are interested in developing rich Web applications that dynamically update content using AJAX and PHP.
- AJAX and scripting Web services with E4X (developerWorks, April 2005): In this series, you find details about AJAX and ECMAScript for XML.
- Ajax for Java developers: Java object serialization for Ajax (developerWorks, October 2005): Learn five ways to serialize data in AJAX applications.
- Build quick, slick Web sites (developerWorks, September 2005): Add XHTML to your Web designs for fast-loading, responsive Web pages.
- The developerWorks Web Architecture zone: Find articles, tutorials, and more about various Web-based solutions.
- Survey AJAX/JavaScript libraries: Visit the OSA Foundation's wiki.
- XUL Planet's object reference section: Get the details on XMLHttpRequest (plus all kinds of other XML objects, as well as DOM, CSS, HTML, Web Service, and Windows and Navigation objects.

Discuss

- Participate in the discussion forum for this content.
- Ajax.NET Professional: Discuss all things AJAX on a great blog.
- developerWorks blogs: Get involved in the developerWorks community..

About the authors

Naveen Balani
Naveen Balani spends most of his time designing and developing J2EE/SOA-based frameworks and products. He has written various articles for IBM developerWorks covering such topics as ESB, Semantic Web, SOA, JMS, AXIS, Web services architectures, CICS, DB2, XML Extender, WebSphere Studio, MQSeries, Java Wireless Devices, and DB2 Everyplace for Palm, J2ME, MIDP, Java-Nokia, Visual Studio .Net, and wireless data synchronization. You can email him at

naveenbalani@rediffmail.com.

Rajeev Hathi
Rajeev Hathi currently works as a Senior Systems Analyst for Satyam Computers Ltd. He spends his time designing and developing J2EE-based frameworks. He likes exploring new technologies and new fields of domains. His pastime hobbies are sports and music. You can reach him at rajeev_hathi@hotmail.com.