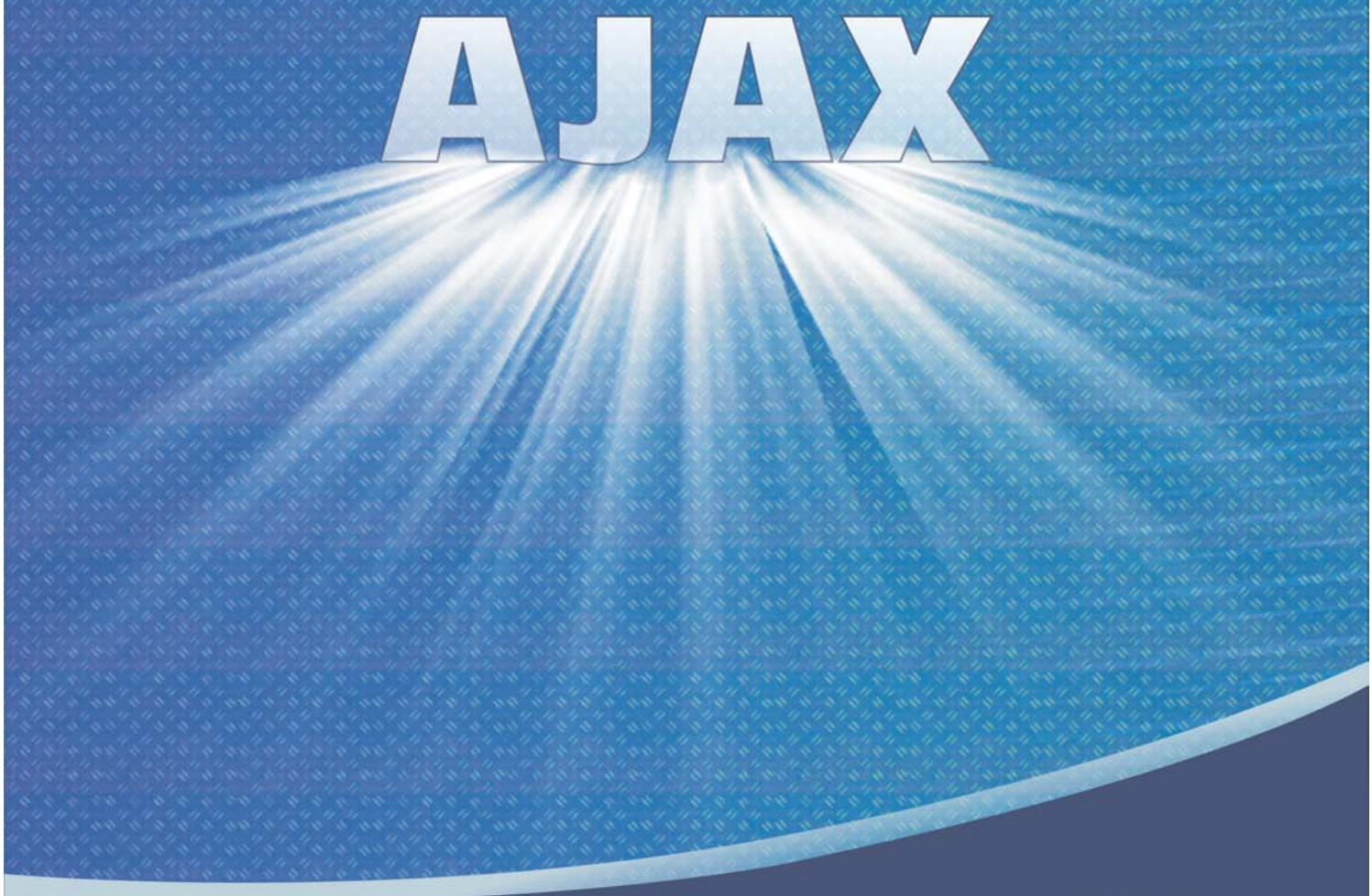
April 08 | Volume 1 |Issue 10 | Free

Java Jazz up

A BETTER WAY TO LEARN PROGRAMMING

AJAX TECHNOLOGY

AJAX WORK



Technical Introduction | Ajax Examples | Advance Ajax | Ajax frameworks types

India's Cheapest web Service Provider

RoseIndia

OUR SERVICES

- ERP Soluiotns
- Software Solutions
- Web Development
- Web Designing
- Web Redesigning
- Domain Registration
- Web Promotion
- SEO
- Article Writing
- Blog Writing
- News Writing
- SEO Copywriting
- Technical Documentation
- E-Commerce Solutions
- CRM
- Outsourcing

Extend your reach with our solutions...

http://www.roseindia.net/services

Java Jazz up
A BETTER WAY TO LEARN PROGRAMMING

April 2008 Volume I Issue X

"Optimism with determination lets you hit the goal harder"

Published by

RoseIndia

Editorial

Dear Readers,

We are back here with the Holi (Mar 2008) issue of Java Jazz-up. The current edition is specially designed for the sprouting technocrats. This issue highlights the interesting Java technologies especially for the beginners.

Though it was a hard job to simplify the complexities of the technologies like Hibernate 3.0, struts 2, JSF and Design Patterns. Still our team has done a marvelous work in making it easy and simpler for the new programmers regime. This issue reflects our consistent attempts to avail the quality technological updates that enforce the readers to appreciate it a lot and be a part of its Readers Community.

Java News and Updates section provides the latest updates of the things happening around the globe making the readers aware of the java technological advancement. In this section, you will know the new features introduced in the existing tools, utilities, application servers, IDEs, along with the Java API updates.

We are providing it in a PDF format so that you can view and even download it as a whole and get its hard copy.

Please send us your feedback about this issue and participate in the Reader's Forum with your problems, issues concerned with the topics you want us to include in our next issues.

Editor-in-Chief
Deepak Kumar
Java Jazz up

Content

Ajax-an introduction

Ajax-an introduction

Developing web application has now been a cup of coffee for developers having the knowledge of Ajax –a technology that was not so easy and hence not popular earlier- now one of the most frequently used technologies. Web application nowadays, as one of the basic needs in terms of fulfilling a small necessity or even more advanced form in the world of Internet, is undoubtedly the best way to convey any message and creation to the world over. Compared to the older generation of Internet the new technologies in the WWW has made it a hassle free work for developers in developing a more interactive, responsive and dynamic site and this has become possible because of Ajax.

Ajax plays a major role in developing fast, interactive, dynamic and responsive site and can run on various operating systems, computer architectures and browsers. Its asynchronous characteristics have made it popular among the users to develop their site using Ajax.

'Rich Internet Application' and 'Web 2.0' usually use this application tool to make site more interactive. The above two web technologies (RIA, Web 2.0) are being used these days to develop popular web trends, 'Social Networking Sites (Orkut, Facebook, MySpace, Ibibo), wikies (Wikipedia) and multimedia content sharing sites (YouTube, Megaupload).

Here, surfers/ viewers will able to access 'the complete' Ajax information. This article is suitable both for beginners and experts.

Ajax sometimes also called the blend of Asynchronous JavaScript and XML is a speedy and quick-responsive way of programming that makes user stress-free. In Ajax, the data, content, and design are merged together to display a perfect picture of programming. It is usually used in new trends of web technologies wikies and content sharing sites.

History in brief

Ajax is only a name that not only reflects any function of the programming but facilitate user works with a pre-existing set of tools to develop the website. Before commencing the name 'Ajax', it was being used as a name of XMLHTTTP in the programming that was inspired by the earlier one with a nearby similar concept XMLhttpRequest.

In the earlier version of Internet Explorer (up to 4.0), a class XMLHttpRequest had been used in JavaScript to make IE browser that was later implemented in the modified form by various web masters to make their own web applications. Google has contributed most to its prominence by implementing XMLHTTP in Gmail and GoogleMaps in 2005.

But the true popularity bursts when XMLHTTP has got its present name 'Ajax'. The 'term' was first used at February 18, 2005 in the web world by Jesse James Garret in his essay called "Ajax: A New Approach to Web Applications". Later on, this technology was immensely used by web companies in developing various technologies like Rich Internet Application, Web 2.0 that were further used to make commonly known websites - 'social networking' and 'wikies'.

Why to use Ajax?

Before interacting to Ajax, let's know first about 'why should we use Ajax despite of having so many programming tools to develop a web application or site?' The reason is simple but strong.

Ajax-an introduction

The functionality of Ajax puts it ahead among the other web tools. Ajax not only builds a fast and dynamic website but also saves the resources. Unlike other web tool, it generates a rich-client application that uses the client's computer for accessing data instead of sole server or network. Earlier the processing of web page used to hold only server-side before sending the whole page within the network.

On the other hand, Ajax can not only modify a portion of the page or even a single field displayed by the browser but can also update it without asking the remaining content to reload. For example: if user changes any field in the form and uploads it, the changes display immediately in the form without showing any error or asking to reload other processed field.

Defining Ajax

Ajax or AJAX is not a single technology but an amalgamation of several technologies that is being used to develop interactive web applications. It is some time referred to Asynchronous Javascript with added XML, which is used to enhance the interactivity, speed, functionality, and usability of the web pages.

Here, the term Asynchronous refers to functionality of the data processing in which user can perform the other request without interfering the backend processing and the desired output displays after ending the page-loading process. For example: if user makes any changes in any form written in Ajax and sends request for changing it through upload the page, the back-end data processing begins and again if during the processing user wants to make another change in the form, then there is no need for the user to wait until the process of page loading finishes. The second made changes would be displayed on the screen.

So, Asynchronous AJAX stands for a combination of techniques that allows web pages to be more interactive and behave like local application that is also know as 'Rich-Client' application. In short, we can define Ajax as an asynchronous powerful technique that integrates:

- Standards-based presentation using XHTML and CSS
- Data interchange with XML and XSLT
- Dynamic display and interaction of pages using through Document Object Model (DOM)
- Asynchronous server communication using XMLHttpRequest
- JavaScript to weave the net among one another

Advantages of Ajax

The biggest advantage of Ajax is its bandwidth usage because it generates HTML locally within the browser and does not require reloading the whole data after amending any portion of the programming. Its speedy response and tendency to share the client's system for accessing content reduces the bandwidth consumption for web applications and server load.

The other foremost advantage is that it supports a wide range of operating system platform, computer architecture, browser and languages.

Disadvantages of Ajax

Despite of having several advantages, Ajax also has some common disadvantages and some of them are:

- Though Ajax is a cross platform tool, yet it has also a limitation, as it cannot run in the older version of Opera and Safari browser and needs to enable XMLHttpRequest to run Ajax. The implementation criteria also differ marginally in all sorts of browsers.

- Ajax generates dynamic pages that do not register itself with cookies, so user can't find previous pages by pressing 'Back' button, but can enable with GET HTTP requests. User also feels to bookmark the earlier pages after updating it and for avoiding this problem; programmer must feed URL fragment identifier. Cookies also don't work with large request and hence need to add the cookie request.

- The IFrame requests in Ajax barred synchronous request and for it, server pages must be designed to work with IFrame request but the implementation process varies in several browser. It can leave extra request in history as per depends up on the browser.

- While making programme, user must consider network latency carefully that can be proved panic for the user with an unusual delay in processing if it is not written properly with correct handling XMLHttp request.

- In the absence of proper sitemap, it was difficult for the Search Engine Optimizer to search the web page as search engine too feel difficulty to execute it, while if it is provided data with full-page refresh.

- As per the latest report, some researchers have found that a large number of power users are not pleased with the performance of Rich Internet Application that was based on Ajax in terms of local rendering of complex business. The rendering process was taking uneven delay.

Ajax Technology

Ajax Technology

Asynchronous JavaScript and XML or Ajax for short is a type of asynchronous programming technical to develop Internet applications. Google made it popular by incorporating the technology in its search engine interface in the year 2005. 'Google Suggest' is the most popular Ajax application.

Ajax is a combination of several technologies that works together to make a web application similar to traditional windows desktop applications. Ajax fills the gap between desktop application and web application. It brings richness of desktop application into Internet world. It is possible to develop highly interactive web applications using Ajax frameworks. The Ajax technologies make it possible to create better, faster, and user-friendlier Internet and Intranet web applications with ease. Examples of Ajax based applications are Gmail and Yahoo Mail.

These days, there is a huge popularity of Rich Internet application in the programming world and companies are looking for Rich Internet applications for their clients. Rich Internet application or RI is nothing but Ajax based application that brings richness and responsiveness to the web applications.

Components of Ajax Technology

Ajax is based on many existing technologies such as CSS, HTML, DHTML and JavaScript, by which programmers are already familiar with in developing web applications.

So, learning Ajax is easy and it requires less time. Let's understand the core components of Ajax Technology. Ajax application uses the following technology in combination:

- **XHTML, HTML and CSS**
  These are used for creating the UI and styling the web pages to make it more appealing.
- **DOM**
  The Document Object Model or DOM is used by the JavaScript code to produce interactive Ajax applications.
- **XMLHttpRequest or XMLHTTP**
  XMLHttpRequest or XMLHTTP is used to retrieve the data from server.
- **Java Script**
  JavaScript is used to bind everything together.

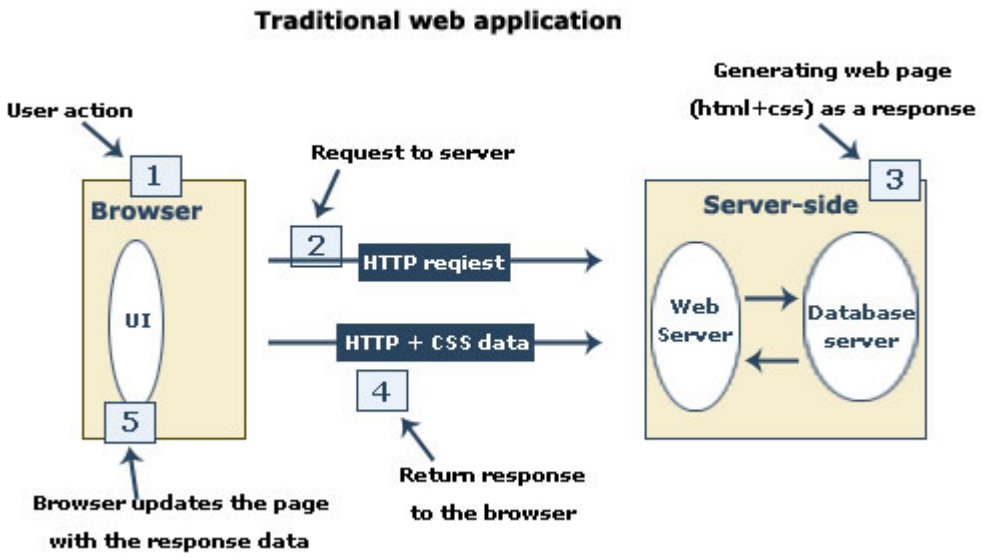
How traditional web application works

How traditional web applications works?

This section explores the working of the browser in traditional web applications and then in the next section we will explore the working of Ajax based application.

Activities involved from making a 'user request from the browser' to 'getting response back to the browser' can be divided into five steps:

- User does something in the browser (For example, User makes a request for a web page)
- Browser sends request for the page to the server
- Server finds the request and generates the requested web page (HTML+CSS) as a response to the request.
- Data is returned in response to the request.
- Now the browser replaces view with the data sent as response from the server.



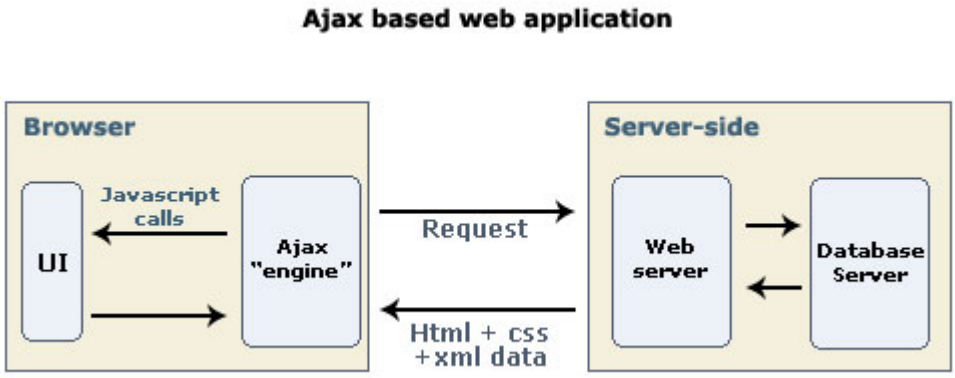
Some points:

1. It all happens only when the user clicks on a link or pushes a button etc.
2. These five steps are involved in each such user interaction.
3. Request processing is completely synchronized with the user driven events i.e. user can now involve with another request after processing the current request and getting response.
4. You can also bookmark the page, move forward and backward.
5. It provides a simple user and browser interaction.

How Ajax Works

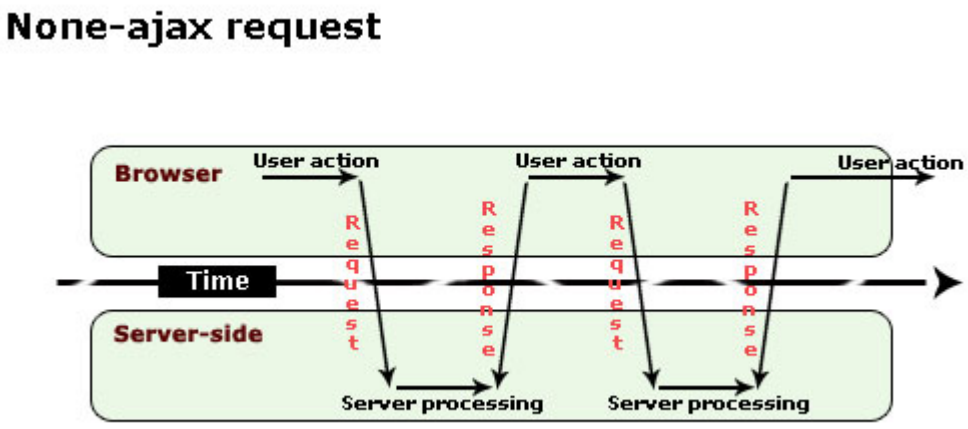
How Ajax Works

Ajax adds an extra layer of functionality in the communication model. Ajax engine acts as an intermediate between the user interaction to the browser and the server system.

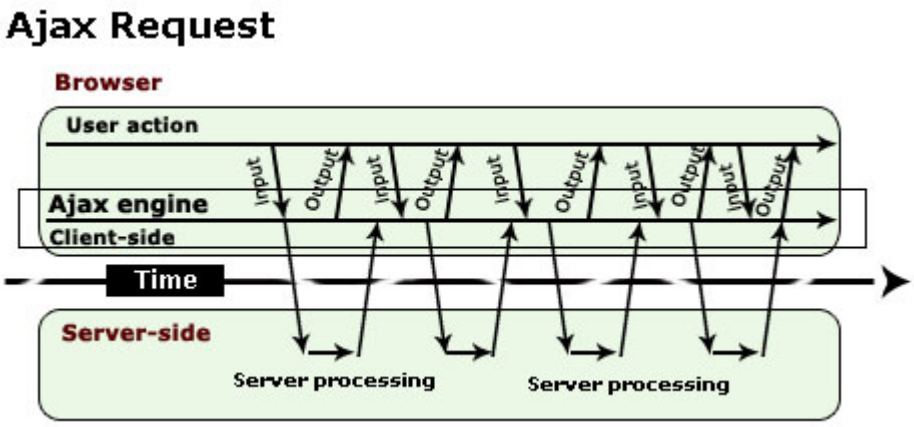


In the traditional web application, the browser communicates the server directly. When the user requests for a page for the first time, the server sends full HTML and CSS code at once. Now if the user makes a new request from the page then the server processes the information, rebuilds the page and sends the full page back to the client browser.

In case of using Ajax, the full page is loaded only once when it is requested first time. Ajax engine, as an intermediate, takes the request for small segment of the page, which then requests information from the web server asynchronously. Here, the word "asynchronously" means that the requested data is collected in the background without interfering with the whole display and behavior of the existing page.

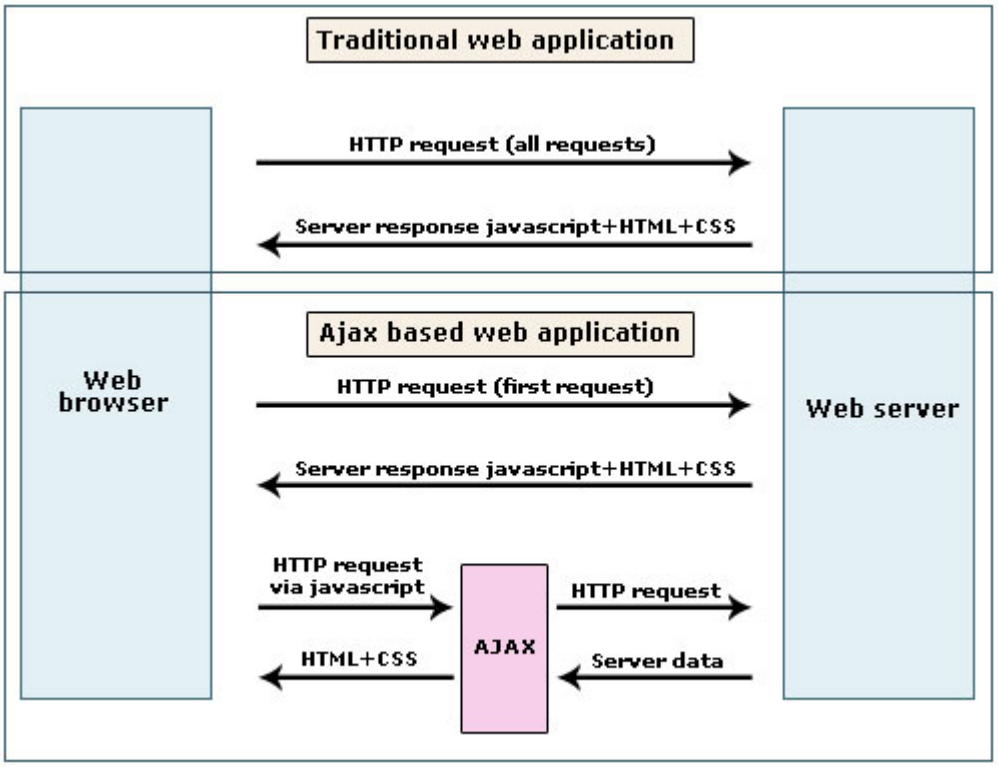


How Ajax Works



Ajax engine does not send the entire page but only the necessary small amount of information to the server. The engine then displays the returned data without reloading the entire page. Ajax uses the JavaScript to asynchronously request and retrieve data from remote servers. Ajax uses XML to collect numerical or text-style data to the browser. It uses JavaScript to extract data from the XML and uses HTML and CSS to display.

The whole process makes the interaction very responsive and creates the feeling of a web application like desktop application because information is displayed immediately.



Ajax - Technical Introduction

Ajax - Technical Introduction

1) Getting the HTTP Request Object:
We have already discussed that Ajax uses JavaScript to make a request from the server.
For this we need an object of such class, which can provide this functionality. For this we have to select any one of the following two classes depending on the browser:

1. ActiveXObject (For Internet Explorer browser) or
2. XMLHttpRequest (For Mozilla, Safari etc.)

To write browser independent code, you can follow the simplified code snippet below:

```
var xmlHttp;
      if (window.XMLHttpRequest) { // Mozilla, Safari, ...
          var xmlHttp = new XMLHttpRequest();
       }
      else if (window.ActiveXObject) { // IE
          var xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
       }
```

2) Deciding the method to be called after receiving the response from the server:
The next step is to decide which JavaScript function is to be called after receiving the response from the server. In this method you can write the code to process and manipulate the data received from the server. There are two ways to set the function for this purpose:

i) Set the property "onreadystatechange" of the Http request object to the name of that JavaScript function
For example, you want JavaScript function "myMethod" to be executed after receiving the response from the server. Then you have to write the code like below:

```
XmlHttp.onreadystatechange = myMethod;   // No bracket after the function.
```

Remember, this line is not for calling the method "myMethod" but only for assigning a reference to the function, which will be called after receiving the response from the server.

ii) Using anonymous function:
Write the code of the function just at the same place instead of referencing the name of the function. For example,

```
xmlHttp.onreadystatechange = function() {
        ……..
        // Do something with the response data
        ……..
}
```

3) Making an HTTP Request
Actually we have not made any request in the previous step. We decided only which method would process the response data after receiving the data from the server. So now it's turn to send the

request to the server.

For this we use two methods of Http request object' class:
i) open()
ii) send()

These methods can be used as given in the code snippet below:

```
xmlHttp.open('POST', 'time.php', true);
xmlHttp.send('name=abc&age=25');
```

open():

This method can take three parameters:

- **First Parameter:**
  Name of HTTP request method (GET, POST, HEAD etc.)
- **Second Parameter:**
  URL of the requested page to read data from.
- **Third Parameter:**
- A boolean value (TRUE, FALSE), to specify whether the request is asynchronous or synchronous.

If it is set to TRUE the request is set as asynchronous i.e. the browser continues the execution of JavaScript function even response has not been received yet from the server.
If it is set to FALSE then request is set to synchronous. In that case the browser waits for the response of the server, which you may not prefer in case of fetching lot of data from the server.

Send():

This method is used to send data to the server with the request. If you don't want to send any then you can write code as below:

```
xmlHttp.send(null)
```

If HTTP request method is of type "POST" then send() method can be used to send any data to the server as param and value pair. For example,

```
"name=abc"
```

You can send as much data to the server by creating the query string like this:

```
"name=abc&age=25"
```

NOTE:
If HTTP request method is of type "POST" then we have to change the MIME type of the request otherwise the server will not accept such data. So, you have to follow the code snippet given below:

xmlHttp.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
4) Working with the Response data:

In the second step, we decided the name of the method "myMethod" to be called after receiving the response from the server. In this method, we write the code handling the response data. For example, myMethod() can have code segment as given below:

```
function myMethod(){
    if (xmlHttp.readyState == 4) {
        if (xmlHttp.status == 200) {
            var str = xmlHttp.responseText;
            alert(str);

        }
        else {
            alert('Request failed');
        }
    }
    else{
        alert ('Request failed');
    }
}
```

i) Checking the state of the request:
Here, readyState property of the HTTP request object "xmlHttp" is used to check the state of the request. This property can have different values indicating the state of the request.

Value	State of the request
0	Un-initialized
1	Loading
2	Loaded
3	Interactive
4	Complete

The method above checks first the state of the request. If it is equal to 4 it means the request has completed and the response is received from the server.

ii) Checking the Status Code of the response:
The next thing that should be checked is the Status Code of the HTTP server response. The server can send different status code depending on the request processing of the server. For example:

Status	Code Definition
200	OK
400	Bad Request
401	Unauthorized
404	Not Found
405	Method Not Allowed
500	Internal Server Error

But for our purpose, status code "200" only is useful. Our method checks and if it is 200 then everything is "OK".

After checking the state and status code we are sure and safe to proceed further. Now we can manipulate the data sent as response from the server.

iii) Accessing the response data:
We have two options to collect the data:
1. **xmlHttp.responseText:** It is used to access plain text response from the server.
2. **xmlHttp.responseXML:** It is used to obtain XML formatted data.

In the method above, the response data is collected in the variable str and then it is used to alert the message.

To dynamically determine the response types you can write:

```
var contentType = xmlHttp.getResponseHeader("content-type");
```

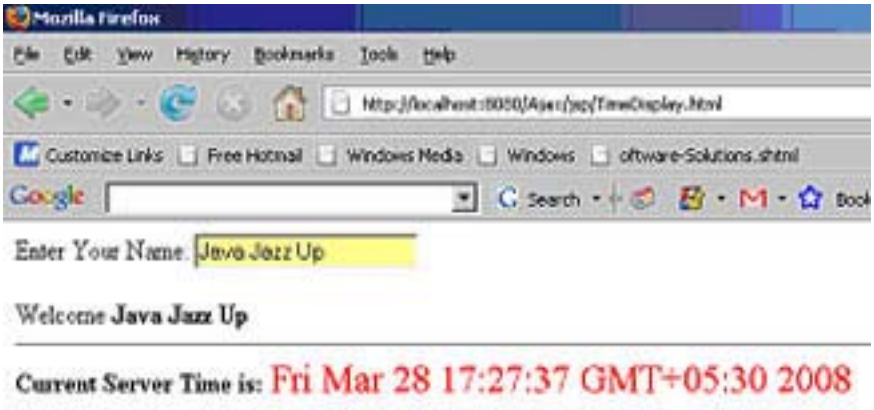
The possible returned content types are 'text/xml' or 'text/plain' indicating XML or plain text content respectively.

Ajax Examples

Ajax Examples

1. Displaying Time:

This example is simple one to understand Ajax with JSP. The objective of the example is to display the current date of the server on the page on each key up event by the user without refreshing the current page. This example also shows the key value entered by the user in the input text field along with the current date and time.



TimeDisplay.html

```
<html>
<body>
<script language="javascript" type="text/javascript">
  function ajaxFunction(){
    var xmlHttp;
    try{
      // Firefox, Opera 8.0, Safari
      xmlHttp=new XMLHttpRequest();
    }
    catch (e){
      // Internet Explorer
      try{
        xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
      }
      catch (e){
        try{
          xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
        }
        catch (e){
          return false;
        }
      }
    }
```

```
    xmlHttp.onreadystatechange=function(){
      if(xmlHttp.readyState==4){
        var name = document.getElementById('name').value;
        document.getElementById('time').innerHTML = "<br/>Welcome <b> "+name+"<hr/
>Current Server Time is:</b> <font color='red' size='5'>" + xmlHttp.responseText + "</
font>";;
      }
    }

    var url = "time.jsp";
    xmlHttp.open("GET",url,true);
    xmlHttp.send(null);
  }
</script>

<form>
  Enter Your Name: <input type="text" onkeyup="ajaxFunction();" name="name" id="name"/>
  <div id="time" ></div>
</form>

</body>
</html>
```

The input text field for user name has "onkeyup" attribute, which is set to the JavaScript function "ajaxFunction()". This method is called every time user leaves the key to move up after pressing the key down. The "div" tag having "id" attribute as "time" is the area where the time is to be displayed.

Ajax code starts from here. "ajaxFunction()" first tries to get HTTP request object maintaining the browser compatibility. In this example, this object is stored in the variable named "xmlHttp". **Read how to get HTTP request object on page 12**

Next step is to determine which method should be invoked after getting the response from the server. In this example method is defined anonymously at the right place. Now the request for the JSP page "time.jsp" is forwarded to the server.

time.jsp:

```
<%@page contentType="text/html" import="java.util.*" %>
<%
response.getWriter().write((new Date()).toString());
%>
```

The server executes the JSP code, which sends current date as a response. After getting the response from the server the anonymous function is executed.

```
if(xmlHttp.readyState==4){
    var name = document.getElementById('name').value;
    document.getElementById('time').innerHTML = "<br/>Welcome <b>
    "+name+"<hr/>Current Server Time is:</b> <font color='red' size='5'>" +
     xmlHttp.responseText + "</font>";
}
```

The function first checks if everything is fine. To get the text response, **responseText** property of the **xmlHttp** object is used. Now the html div component is populated with the welcome string and the response text.

2. Upldating First Name and Last Name using Ajax:

This example aims to update the first and last name of the user without refreshing the current page. The user fills the user number and when looses focus from the input component, its first and last names are updated. Put user numbers "1" and "2" only to show the results in this example.

UserInfo.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
 <title>Ajax and Jsp, Updating First Name and Last Name using Ajax</title>
 <script language="javascript" type="text/javascript">
 var xmlHttp = getHTTPRequestObject();
 function showFirstLastName() {
   xmlHttp.onreadystatechange = updatepage;

   var userNumber = document.getElementById("userNo").value;
   var url = "user.jsp?userNo="+ userNumber;
   xmlHttp.open("GET", url, true);
   xmlHttp.send(null);
 }

 function updatepage() {
   if (xmlHttp.readyState == 4) {
     if (xmlHttp.status == 200) {
       // For XML formatted response
            var message = xmlHttp.responseXML.getElementsByTagName("name")[0];
            results = message.childNodes[0].nodeValue.split(",");
            document.getElementById('firstName').value = results[0];
            document.getElementById('lastName').value = results[1];
```

```
    }
   }
  }

  function getHTTPRequestObject() {
    var xmlhttp;
    if (window.XMLHttpRequest) {
      xmlhttp = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
      xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
    }
    return xmlhttp;
  }
</script>

</head>
<body>
<form>
  <p>Enter User Number:
  <input size="10" name="User No" id="userNo" type="text" onblur="showFirstLastName();"></
p>
  First Name:
  <input size="10" name="First Name" id="firstName" type="text">
  Last Name:
  <input size="10" name="Last Name" id="lastName" type="text"></form>
</body>
</html>
```

The input text field for user number has "onblur" attribute, which is set to the JavaScript function "showFirstLastName ()". This method is called every time user looses focus from the component. To show the first and last name there are two more input components of id "firstName" and "lastName" which will be updated after getting the records corresponding to the particular user using Ajax.

"showFirstLastName ()" first tries to get HTTP request object maintaining the browser compatibility. In this example, this object is stored in the variable named "xmlHttp". **Read how to get HTTP request on page 12**

Next step is to determine which method should be invoked after getting the response from the server. In this example, the method "updatepage()" is assigned for this purpose. Now the request for the JSP page "user.jsp" is forwarded to the server along with user number as parameter

user.jsp

```
<%
String userNumber = request.getParameter("userNo");
if(userNumber != null) {
  response.setContentType("text/xml");
  response.setHeader("Cache-Control", "no-cache");
       // For XML formatted message
    if(userNumber.equals("1")){
      response.getWriter().write("<name>Deepak,Kumar</name>");
    }
```

```
    else if(userNumber.equals("2")){
      response.getWriter().write("<name>Ravi,Kant</name>");
    }
    else{
      response.getWriter().write("<name>,</name>");
    }
} else {
  //set Status code 204 (SC_NO_CONTENT) indicating that the request succeeded but no new
information to return.
  response.setStatus(HttpServletResponse.SC_NO_CONTENT);
}
%>
```

The server executes the JSP code, which sends first and last name in xml format. First and last names are separated with comma and are enclosed within <name> element. After getting the response, method "updatepage()" is called to manipulate the returned data.

```
if (xmlHttp.readyState == 4) {
       if (xmlHttp.status == 200) {
              // For XML formatted response
              var message = xmlHttp.responseXML.getElementsByTagName("name")[0];
              results = message.childNodes[0].nodeValue.split(",");
              document.getElementById('firstName').value = results[0];
              document.getElementById('lastName').value = results[1];
  }
}
```

The function first checks if everything is fine. To get the xml response, **responseXML** property of the **xmlHttp** object is used. Get the nodes values of xml response and split it with comma. Now splitted values are set to the input components of id "**firstName**" and "**lastName**". First name and last names of the user are updated and so displayed on the page.

3. Checking User Name Availability in database

This example checks the user name availability using Ajax. If user name does not exist then it congratulates the user otherwise it warns the user to choose any other name.

AjaxUserNameAvailability.html

Ajax Examples

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
 <head>
  <title>AJAX and Servlet</title>

 <script language="javascript">
   var xmlHttp;
   function checkUserExist() {
     var username = document.getElementById("username").value;
     var url = "/Ajax/AJAXCheckUserServlet?username=" + username;
     if(window.XMLHttpRequest){
       xmlHttp = new XMLHttpRequest();
     }
     else if(window.ActiveXObject){
       xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
     }

     xmlHttp.open("Get",url,true);
     xmlHttp.onreadystatechange = callback;
     xmlHttp.send(null);
   }

   function callback() {
     if( xmlHttp.readyState==4 ){
       if( xmlHttp.status==200 ) {
         document.getElementById('text').innerHTML = "<br/><b>"+xmlHttp.responseText+"</
b>";
       }
     }
   }

   function focusIn(){
     document.getElementById("username").focus( );
   }
 </script>
 </head>

<body onload="focusIn();">
 <form>
   User Name <input type="text" id="username" name="username" />
   <input type="button" value="Check User Name Availability" onClick="checkUserExist()"/>
   <div id=" text" ></div>
 </form>

</body>
</html>
```

Traditionally, when the user registers itself to create the account with the application, there may be many fields to be entered in addition to the user name. Now the whole data is submitted to the server. If the user name with the same name exists already then application sends back the same page again to let the user choose any other user name.

Ajax Examples

With the use of Ajax, transferring of large amount of data to the server can be minimized by sending only user name to the server where user name existence can be checked.

In the above code, there is a button component with label "**Check User Name Availability**". It has "**onClick**" attribute which is set to the JavaScript function "**checkUserExist()**". This method is called every time user clicks the button component.
div area of id "**text**" is used to show the result of the response from the server using Ajax.

"**checkUserExist()**" first tries to get HTTP request object maintaining the browser compatibility. In this example, this object is stored in the variable named "xmlHttp". **Read how to get Http request object on page 12**

Next step is to determine which method should be invoked after getting the response from the server. In this example, the method "**callback()**" is assigned for this purpose. Now the request for the servlet "**AJAXCheckUserServlet**" is forwarded to the server along with user name as parameter.

AJAXCheckUserServlet.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.sql.*;

public class AJAXCheckUserServlet extends HttpServlet {

  public void doGet( HttpServletRequest request, HttpServletResponse response ) throws
ServletException, IOException {

    String username = request.getParameter("username");
    boolean isExist = check(username);

    response.setContentType("text/html");
    response.setHeader("Cache-Control", "no-cache");

    if(isExist){
      response.getWriter().write("User Name exists. Please select other name.");
    }
    else{
      response.getWriter().write("Congratulations....User Name is available.");
    }
  }

  public boolean check(String username) {
    Connection con = null;
    String url = "jdbc:mysql://192.168.10.59:3306/";
    String db = "javajazzup";
    String driver = "com.mysql.jdbc.Driver";
    String user = "root";
    String pass = "root";
    boolean isExist= false;
    try{
      Class.forName(driver).newInstance();
```

```
      con = DriverManager.getConnection(url+db, user, pass);
      try{
      Statement st = con.createStatement();
      ResultSet res = st.executeQuery("SELECT user_name FROM  users");

      while (res.next()) {
        String un = res.getString("user_name");
       if(username.equals(un)){
          con.close();
          isExist = true;
          break;
        }
      }
      con.close();

      }
      catch (SQLException s){
      System.out.println("SQL code does not execute.");
      }
    }
    catch (Exception e){
     e.printStackTrace();
    }
    return isExist;
  }

}
```

This servlet finds the username parameter and checks if this username is available in the database. If it exists then sends **"User Name exists. Please select other name."** otherwise **"Congratulations....User Name is available."** to the user. After getting the response from the server JavaScript function **"callback()"** is called to handle the response data.

```
if( xmlHttp.readyState==4 ){
      if( xmlHttp.status==200 ) {
          document.getElementById('text').innerHTML =
           "<br/><b>"+xmlHttp.responseText+"</b>";
      }
}
```

The function first checks if everything is fine. To get the text response, **responseText** property of the **xmlHttp** object is used. Now the html div component of id **"text"** is set to the response text.

4. Show User Details:

This Ajax example shows how to show user details like first and last name of the user without refreshing the current page. When the user changes user selection, its first and last names are displayed on the page. JavaScript is used to extract the XML data of the response.

Ajax Examples

AjaxXml.html

```
<html>
<head><title>Ajax and Servlet Using XML</title>
<script language="javascript">
    var xmlHttp;
    function showUser(username) {
      var url = "/Ajax/AJAXUserInfoServlet?username=" + username;
      if(window.XMLHttpRequest){
        xmlHttp = new XMLHttpRequest();
      }
      else if(window.ActiveXObject){
        xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
      }

      xmlHttp.open("Get",url,true);
      xmlHttp.onreadystatechange = callback;
      xmlHttp.send(null);
    }
function callback() {
      if( xmlHttp.readyState==4 ){
        if( xmlHttp.status==200 ) {
          xmlDoc=xmlHttp.responseXML;
          document.getElementById("firstname").innerHTML=
xmlDoc.getElementsByTagName("first_name")[0].childNodes[0].nodeValue;
          document.getElementById("lastname").innerHTML=
xmlDoc.getElementsByTagName("last_name")[0].childNodes[0].nodeValue;

        }
      }
    }
  </script>
  </head>
<body>
<form>
<b>Select a User:</b>
<select name="users" onchange="showUser(this.value)">
```

```
<option value=”deepak1">deepak1</option>
<option value=”deepak2">deepak2</option>
</select>
</form>

<h3>
  <span id=”firstname”></span>  
  <span id=”lastname”></span>
</h3>

</body>
</html>
```

The combo box, which is used to select one option, has **“onchange”** attribute, which is set to the JavaScript function “**showUser ()**”. This method is called every time user changes the user selection. To show the user details two components of id “firstname” and “lastname” is used which will be updated after getting the records corresponding to the particular user using Ajax.

“**showUser ()**” first tries to get HTTP request object maintaining the browser compatibility. In this example, this object is stored in the variable named “xmlHttp”. **Read how to get Http request object on page 12**

Next step is to determine which method should be invoked after getting the response from the server. In this example, the method “**callback ()**” is assigned for this purpose. Now the request for servlet “**AJAXUserInfoServlet**” is forwarded to the server along with user number as parameter.

AJAXUserInfoServlet.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.sql.*;

 public class AJAXUserInfoServlet extends HttpServlet {

  public void doGet( HttpServletRequest request, HttpServletResponse response ) throws
ServletException, IOException {

    String username = request.getParameter(“username”);
    response.setContentType(“text/xml”);
    response.setHeader(“Cache-Control”, “no-cache”);

    sendUserFirstLastName(username,response);
  }

  public void sendUserFirstLastName(String username,HttpServletResponse response) {
    Connection con = null;
    String url = “jdbc:mysql://192.168.10.59:3306/”;
    String db = “javajazzup”;
    String driver = “com.mysql.jdbc.Driver”;
    String user = “root”;
```

```
    String pass = "root";

    try{
     Class.forName(driver).newInstance();
      con = DriverManager.getConnection(url+db, user, pass);
      try{
      Statement st = con.createStatement();
      ResultSet res = st.executeQuery("SELECT * FROM  users where
user_name='"+username+"'");
      StringBuffer responseXML = new StringBuffer();
      responseXML.append("<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?>");
      responseXML.append("<user>");
      while (res.next()) {
        String un = res.getString("user_name");
        String fn = res.getString("first_name");
        String ln = res.getString("last_name");

       responseXML.append("<user_name>"+un+"</user_name>");
       responseXML.append("<first_name>"+fn+"</first_name>");
       responseXML.append("<last_name>"+ln+"</last_name>");

      }
      responseXML.append("</user>");
      response.getWriter().write(responseXML.toString());
      con.close();

      }
      catch (SQLException s){
      System.out.println("SQL code does not execute.");
      }
    }
    catch (Exception e){
     e.printStackTrace();
    }

  }

}
```

This servlet sends the response in XML format. Now after getting the response back the JavaScript function "**callback ()**" is called to work on server response data.

```
if( xmlHttp.readyState==4 ){
    if( xmlHttp.status==200 ) {
          xmlDoc=xmlHttp.responseXML;

  document.getElementById("firstname").innerHTML=
xmlDoc.getElementsByTagName("first_name")[0].childNodes[0].nodeValue;
          document.getElementById("lastname").innerHTML=
xmlDoc.getElementsByTagName("last_name")[0].childNodes[0].nodeValue;
     }
}
```

The function first checks if everything is fine. To get the xml response, **responseXML** property of the **xmlHttp** object is used. Get the nodes values of xml response. These values are set to the components of id "**firstname**" and "**lastname**". First name and last names of the user are updated and so displayed on the page.

5. Display Number in Words

This example displays a number in words. For example, putting number "1" is displayed as word "One". This example converts numbers ranging from 0 to 9. This conversion is displayed using Ajax, which does not refresh the page at all but only displays only the equivalent converted string for the number.

AjaxNumberToWord.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
 <head>
  <title>AJAX and Servlet</title>
  <script language="javascript">
   var req;
   function convertToString() {
     var num = document.getElementById("num");
     var url = "/Ajax/AJAXResponseServlet?num=" + escape(num.value);
     if(window.XMLHttpRequest){
       req = new XMLHttpRequest();
     }
     else if(window.ActiveXObject){
       req = new ActiveXObject("Microsoft.XMLHTTP");
     }

     req.open("Get",url,true);
     req.onreadystatechange = callback;
     req.send(null);
   }

   function callback() {
     if( req.readyState==4 ){
       if( req.status==200 ) {
```

```
          var strNum = document.getElementById("numstring");
          strNum.value = req.responseText;
        }
      }
    }

    function clear(){
      var num = document.getElementById("num");
      num.value = "";
    }

    function focusIn(){
      document.getElementById("num").focus( );
    }
  </script>
 </head>
  <body onload="focusIn();">
    Enter the Number here: <input type="text" id="num" name="num"
onkeyup="convertToString();">
    <br><br>
    Equivalent String: <input type="text" size="20" readonly id="numstring">

  </body>
</html>
```

The above html page has input text field of id "num" with "onkeyup" attribute that is set to the JavaScript function "**convertToString()**". This method is called each time the user release the key up.

"**convertToString ()**" first tries to get HTTP request object maintaining the browser compatibility. In this example, this object is stored in the variable named "xmlHttp". **Read how to get Http request object on page 12**

Next step is to determine which method should be invoked after getting the response from the server. In this example, the method "**callback ()**" is assigned for this purpose. Now the request for servlet "**AJAXResponseServlet**" is forwarded to the server along with number as parameter.

AJAXResponseServlet.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class AJAXResponseServlet extends HttpServlet {

  public void doGet( HttpServletRequest request, HttpServletResponse response ) throws
ServletException, IOException {

    String number = request.getParameter("num");
    String strRepr = null;
```

```
    if(!(number.trim().equals("”))){
      switch(Integer.parseInt(number)){
        case 0:
          strRepr = “Zero”;
        break;
        case 1:
          strRepr = “One”;
        break;
        case 2:
          strRepr = “Two”;
        break;
        case 3:
          strRepr = “Three”;
        break;
        case 4:
          strRepr = “Four”;
        break;
        case 5:
          strRepr = “Five”;
        break;
        case 6:
          strRepr = “Six”;
        break;
        case 7:
          strRepr = “Seven”;
        break;
        case 8:
          strRepr = “Eight”;
        break;
        case 9:
          strRepr = “Nine”;
        break;
        default:
          strRepr = “Sorry, no conversion available for the number “+number+”.”;
      }

      //Set up the response
      response.setContentType(“text/html”);
      response.setHeader(“Cache-Control”, “no-cache”);
      response.getWriter().write(strRepr);
      }
      else{
        response.setContentType(“text/html”);
        response.setHeader(“Cache-Control”, “no-cache”);
        response.getWriter().write(“?”);
      }
  }
}
```

This servlet sends the response to the client. Now after getting the response back the JavaScript function “**callback ()**” is called to work on server response data.

```
if( req.readyState==4 ){
     if( req.status==200 ) {
          var strNum = document.getElementById("numstring");
           strNum.value = req.responseText;
     }
}
```

The function first checks if everything is fine. To get the text response, **responseText** property of the **xmlHttp** object is used. This value is set to the components of id "**numstring**". This updated component is displayed on the page.

6. Ajax Conversion:

This example converts a number into Binary, Octal, Decimal, Hexadecimal formats using Ajax without refreshing the whole page.

AJAXConverterUsingXML.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
 <head>
  <title>AJAX Number Converter</title>
  <script language="javascript">
 var request;
 function convert( ){
   var number = document.getElementById("number");
   var url = "/ajaxapp/AjaxNumberConverterServlet?number=" + number.value;
   if(window.XMLHttpRequest){
```

```
    request = new XMLHttpRequest();
  } else{
    request = new ActiveXObject("Microsoft.XMLHTTP");
  }
  request.open("Get",url,true);
  request.onreadystatechange = handleResponse;
  request.send(null);
}

function handleResponse( ){
  if(request.readyState == 4){
    if(request.status == 200){
      if(window.XMLHttpRequest){
        nonMSPopulate();
      } else{
        msPopulate();
      }
    }
  }
}

function nonMSPopulate( ){

  var response = request.responseText;

  var parser = new DOMParser();
  var dom = parser.parseFromString(response,"text/xml");

  binVal = dom.getElementsByTagName("binary");
  var binary = document.getElementById("binary");
  binary.value = binVal[0].childNodes[0].nodeValue;

  octVal = dom.getElementsByTagName("octal");
  var octal = document.getElementById("octal");
  octal.value = octVal[0].childNodes[0].nodeValue;

  decVal = dom.getElementsByTagName("decimal");
  var decimal = document.getElementById("decimal");
  decimal.value = decVal[0].childNodes[0].nodeValue;

  hexVal = dom.getElementsByTagName("hexadecimal");
  var hexadecimal = document.getElementById("hexadecimal");
  hexadecimal.value = hexVal[0].childNodes[0].nodeValue;
}

function msPopulate( ){

  var response = request.responseText;

  var xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
  xmlDoc.async = "false";
```

```
    xmlDoc.loadXML(response);

    bin = xmlDoc.getElementsByTagName('binary');
    var binary = document.getElementById('binary');
    binary.value = bin[0].firstChild.data;

    oct = xmlDoc.getElementsByTagName('octal');
    var octal = document.getElementById('octal');
    octal.value = oct[0].firstChild.data;

    dec = xmlDoc.getElementsByTagName('decimal');
    var decimal = document.getElementById('decimal');
    decimal.value = dec[0].firstChild.data;

    hex = xmlDoc.getElementsByTagName('hexadecimal');
    var hexadecimal = document.getElementById('hexadecimal');
    hexadecimal.value = hex[0].firstChild.data;
  }

  function focusIn( ){
    document.getElementById("number").focus;
  }

    </script>
  </head>

  <body onload="focusIn( );">
    <h1>AJAX Conversion</h1> <br><br>

    <table>
  <tr>
    <td>
      Enter Number here:   <input type="text" id="number" name="number"
maxlength="2" size="2" onkeyup="convert( );">
    </td>
  </tr>
   </table>

    <table border="1">
  <tr>
    <td align="center" colspan="5">
      <b>Converted Values</b>
    </td>
  </tr>
  <tr>
    <td align="center">Binary</td>
    <td align="center">Octal</td>
    <td align="center">Decimal</td>
    <td align="center">Hexadecimal</td>
  </tr>
  <tr>
```

```
    <td align="center">
      <input type="text" readonly id="binary">
    </td>
    <td align="center">
      <input type="text" readonly id="octal">
    </td>
    <td align="center">
      <input type="text" readonly id="decimal">
    </td>
    <td align="center">
      <input type="text" readonly id="hexadecimal">
    </td>
  </tr>
   </table>

   </body>
</html>
```

The above html page has input text field of id "number" with "onkeyup" attribute that is set to the JavaScript function "**convert ()**". This method is called each time the user release the key up.

"**convert ()**" first tries to get HTTP request object maintaining the browser compatibility. In this example, this object is stored in the variable named "xmlHttp". **Read how to get Http request object on page 12**

Next step is to determine which method should be invoked after getting the response from the server. In this example, the method "**handleResponse ()**" is assigned for this purpose. Now the request for servlet "**AjaxNumberConverterServlet**" is forwarded to the server along with number as parameter.

AjaxNumberConverterServlet.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class AjaxNumberConverterServlet extends HttpServlet {

  public void doGet( HttpServletRequest request, HttpServletResponse response ) throws
ServletException, IOException {

    String strEnteredKey = request.getParameter("number");
    StringBuffer responseXML = null;

    if( strEnteredKey!="" ){
      int num = Integer.parseInt(strEnteredKey);
      responseXML = new StringBuffer("\r\n<converted-values>");
      responseXML.append("\r\n<binary>"+Integer.toBinaryString(num)+"</binary>");
      responseXML.append("\r\n<octal>"+Integer.toString(num,8)+"</octal>");
      responseXML.append("\r\n<decimal>"+num+"</decimal>");
      responseXML.append("\r\n<hexadecimal>"+Integer.toString(num,16)+"</hexadecimal>");
```

```
      responseXML.append("</converted-values>");
      System.out.println("\n"+responseXML.toString());
      response.setContentType("text/xml");
      response.setHeader("Cache-Control", "no-cache");
      response.getWriter().write(responseXML.toString());
    }
    else{
      responseXML = new StringBuffer("\r\n<converted-values>");
      responseXML.append("\r\n<binary>?</binary>");
      responseXML.append("\r\n<octal>?</octal>");
      responseXML.append("\r\n<decimal>?</decimal>");
      responseXML.append("\r\n<hexadecimal>?</hexadecimal>");
      responseXML.append("</converted-values>");
      System.out.println("\n"+responseXML.toString());
      response.setContentType("text/xml");
      response.setHeader("Cache-Control", "no-cache");
      response.getWriter().write(responseXML.toString());
    }
  }
}
```

The server executes the servlet code, which sends the data in xml format. After getting the response, method "**handleResponse** ()" is called to manipulate the returned data.

```
if(request.readyState == 4){
        if(request.status == 200){
              if(window.XMLHttpRequest){
                    nonMSPopulate();
              } else{
    msPopulate();
                }
  }
}
```

This function first checks if everything is fine. The above code checks if it is Microsoft's browser and call **msPopulate()** method otherwise **nonMSPopulate()** method is called. This is done because Microsoft's XML parser id different than others.

All browsers have built-in XML parser to read and manipulate XML. It reads XML in memory and converts into XML DOM objects. Now these objects can be accessed with JavaScript.

The following JavaScript fragment loads the response data into the parser:

For Non-Microsoft browsers:

```
var parser = new DOMParser();
var dom = parser.parseFromString(response,"text/xml");
```

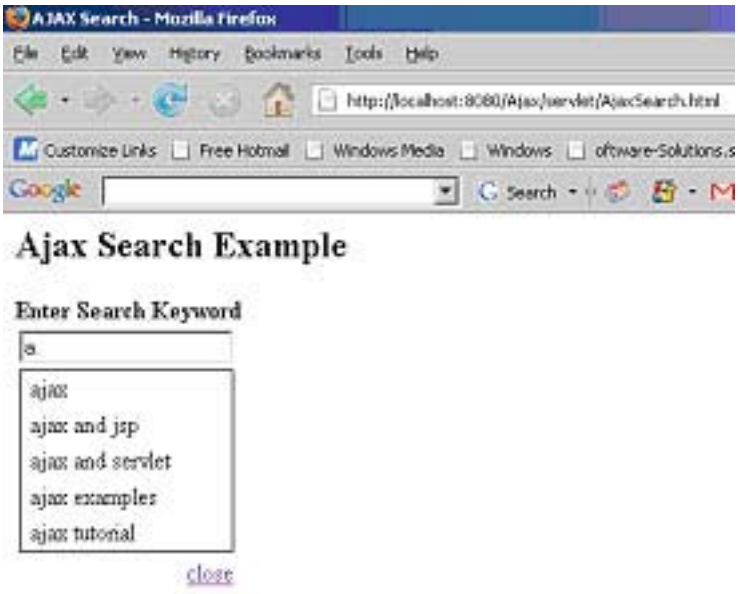
For Microsoft browser:

```
var xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
```

```
xmlDoc.async = "false";
xmlDoc.loadXML(response);
```

Extract values from xml data and set to the appropriate html component. These values will be visible on the page.

7. Ajax Search:
This example searches the user keyword in the database and shows the matching keywords on the page.



AjaxSearch.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
 <head>
  <title>AJAX Search</title>
   <style type="text/css">
   .normal {
     background-color: #FFFFFF;
     padding: 2px 6px 2px 6px;
   }
   .over {
     background-color: #3366CC;
     padding: 2px 6px 2px 6px;
   }
   div#result {
    display: none;
```

```
      border:1px solid #000000;
    }

  </style>
  <script language="javascript">
    var xmlHttp;
    function searchText() {
      var search = document.getElementById("search").value;
      var url = "/Ajax/AJAXSearchServlet?search=" + search;
if(window.XMLHttpRequest){
        xmlHttp = new XMLHttpRequest();
      }
      else if(window.ActiveXObject){
        xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
      }

      xmlHttp.open("Get",url,true);
      xmlHttp.onreadystatechange = callback;
      xmlHttp.send(null);
    }

    function callback() {
      if( xmlHttp.readyState==4 ){
        if( xmlHttp.status==200 ) {

          var result = document.getElementById('result');
          result.innerHTML = '';
          var respText = xmlHttp.responseText;
          if((respText.length)!=0){
            show_div('result');
            show_div('close');

            var str = xmlHttp.responseText.split("\n");
            var items;
            for(i=0; i < str.length - 1; i++) {
              items = '<div onmouseover="javascript:overText(this);" ';
              items += 'onmouseout="javascript:outText(this);" ';
              items += 'onclick="javascript:setText(this.innerHTML);" ';
              items += 'class="normal">' + str[i] + '</div>';
              result.innerHTML += items;
            }
          }
          else
          {
            hide_divs();
          }

        }
      }
    }
```

```
    function focusIn(){
      document.getElementById("search").focus( );
    }

    function overText(div_value) {
      div_value.className = 'over';
    }

    function outText(div_value) {
div_value.className = 'normal';
    }

    function setText(value) {
      document.getElementById('search').value = value;
      hide_divs();
      document.getElementById('result').innerHTML = '';
    }

    function show_div(div_id) {
      document.getElementById(div_id).style.display = 'block';
    }

    function hide_divs() {
      document.getElementById('result').style.display = 'none';
      document.getElementById('close').style.display = 'none';
    }
  </script>

  </head>

  <body onload="focusIn();">
    <h2>Ajax Search Example</h2>

    <form>
      <b>Enter Search Keyword</b>
      <table border = "0">
        <tr>
          <td><input type="text" id="search" name="search" onkeyup="searchText()"
autocomplete="off" /></td>
        </tr>
        <tr>
          <td><div id="result"></div></td>
        </tr>
        <tr>
          <td><div id="close" align="right" style="display: none;"><a href="" onclick="hide_divs();
return false">close</a></div></td>
        </tr>
      </table>
    </form>
  </body>
</html>
```

Ajax Examples

The above html page has input text field of id "**search**" with "**onkeyup**" attribute that is set to the JavaScript function "**searchText ()**". This method is called each time the user releases the key up. This input text component also has an attribute "**autocomplete**" which is set to the value "**off**". This attribute is used to turn on/off the default behavior of text field to show texts entered before. But here we have to show matched data from the database. So it is necessary to set this attribute to 'off'.

"**searchText ()**" first tries to get HTTP request object maintaining the browser compatibility. In this example, this object is stored in the variable named "xmlHttp". **Read how to get Http request object on page 12**

Next step is to determine which method should be invoked after getting the response from the server. In this example, the method "**callback ()**" is assigned for this purpose. Now the request for servlet "**AJAXSearchServlet**" is forwarded to the server along with search keyword as parameter.

AJAXSearchServlet.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.sql.*;

public class AJAXSearchServlet extends HttpServlet {

  public void doGet( HttpServletRequest request, HttpServletResponse response ) throws
ServletException, IOException {

    String search = request.getParameter("search");

    response.setContentType("text/html");
    response.setHeader("Cache-Control", "no-cache");
    if(!((search.trim()).equals(""))){
      String searchString = getSearchResult(search);
      System.out.println(searchString);
      response.getWriter().write(searchString);
    }
    else{
      System.out.println("Length of string"+("".length()));
      response.getWriter().write("");
    }
  }

  public String getSearchResult(String search) {
    Connection con = null;
    String url = "jdbc:mysql://192.168.10.59:3306/";
    String db = "javajazzup";
    String driver = "com.mysql.jdbc.Driver";
    String user = "root";
    String pass = "root";
    String finalSearch="";
    try{
```

```
     Class.forName(driver).newInstance();
      con = DriverManager.getConnection(url+db, user, pass);
      try{
      Statement st = con.createStatement();
      ResultSet res = st.executeQuery("SELECT distinct(keyword_name) FROM keywords WHERE
keyword_name like('"+search+"%') ORDER BY keyword_name");

      while (res.next()) {
        String un = res.getString("keyword_name");
       finalSearch+= un+"\n";
      }
      con.close();

      }
      catch (SQLException s){
      System.out.println("SQL code does not execute.");
      }
    }
    catch (Exception e){
     e.printStackTrace();
    }
    return finalSearch;
  }

}
```

This servlet sends the matched keywords separated with "\n". After receiving the response from the server the JavaScript function "**callback()**" is called.

```
if( xmlHttp.readyState==4 ){
  if( xmlHttp.status==200 ) {
    var result = document.getElementById('result');
    result.innerHTML = '';
    var respText = xmlHttp.responseText;
    if((respText.length)!=0){
      show_div('result');
      show_div('close');
      var str = xmlHttp.responseText.split("\n");
      var items;
      for(i=0; i < str.length - 1; i++) {
        items = '<div onmouseover="javascript:overText(this);" ';
        items += 'onmouseout="javascript:outText(this);" ';
        items += 'onclick="javascript:setText(this.innerHTML);" ';
        items += 'class="normal">' + str[i] + '</div>';
        result.innerHTML += items;
      }
    }
    else{
                          hide_divs();
    }
  }
}
```

The function first checks if everything is fine. To get the text response, **responseText** property of the **xmlHttp** object is used. Split the response with "\n" and put each separated value in a div component. Now all div components are attached with the div component of id "**result**". This updated component is displayed on the page. Different JavaScript functions are used to show and hide components on different events.

Ajax Examples

1. Hello Ajax World Example

The objective of this example is to display the content of "data.xml" file when the button on the page is clicked. Clicking the button doesn't refresh the page. The above page contains button component containing "onclick" attribute which is set to JavaScript function "replace()". This function is called each time this button is clicked. The "div" component of "message" id is used to display the time when Ajax response is completed.

helloworld.html

```
<Html>
<Head>
<Title>Simple Ajax and XML Example</title>
<script type="text/javascript">
function replace(){
var xmlHttp;
if (window.XMLHttpRequest){ // Mozilla, Safari, ...
var xmlHttp = new XMLHttpRequest();
}
else if (window.ActiveXObject) { // IE
var xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
}

xmlHttp.onreadystatechange=function() {
if(xmlHttp.readyState == 4) {
process();
}
}
xmlHttp.open("GET", "data.xml");
xmlHttp.send(null);
}


function  process(){
var message = xmlHttp.responseXML.getElementsByTagName("display-name")[0];
var results = message.childNodes[0].nodeValue;
document.getElementById('message').innerHTML=results;
}
</script>
</head>
<body>
<center>
<table border="0" cellpadding="0" cellspacing="1" width="464" bgcolor="#ccffcc">
<tr>
<td width="525" colspan="2" ><font color="blue">
<p align="center"><b>Hello Ajax World - Example</b></font></td>
</tr>
<tr>
<td width="289" ><font color="blue"> Click to see Message!</font></td>
</tr>
<tr>
```

```
<td><input type="button" onclick="javascript:replace()" value="Get Message!"></a></td>
</tr>
<tr>
<td><div id="message"></div></td>
</tr>
</table>
</center>
</body>
</html>
</html>
```

JavaScript code first tries to get HTTP request object maintaining the browser compatibility. In this example, this object is stored in the variable named "xmlHttp". replace() method extracts values from xml file and set it to the div component of "message" id.2. data.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app>
<display-name>Hello Ajax World!</display-name>
</web-app>
```

Now save both the file as helloworld.html and data.xml and run it on your localhost server it will look like same as given image1

Now click on "Get Message!" button and you will see the result same as given in image2

This was the basic "Hello Ajax World!" example, now run some more examples given in this issue for better understanding.

Ajax Examples

2. Show Time using Ajax and php

The objective of this example is to show current server time when the button on the page is clicked. Clicking the button doesn't refresh the page. The above page contains button component containing "onclick" attribute which is set to JavaScript function "showCurrentTime()". This function is called each time this button is clicked. The “div” component of “date” id is used to display the time when Ajax response is completed.showtime.html

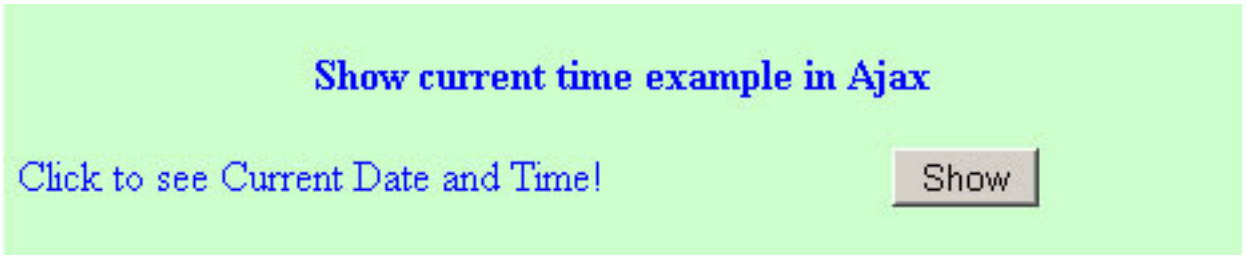
```
<html>
<head>

<title>Ajax Example</title>

<script language="Javascript">

function  postRequest(strURL)  {

var xmlHttp;

if (window.XMLHttpRequest) { // Mozilla, Safari, ...

var xmlHttp = new XMLHttpRequest();

} else if (window.ActiveXObject) { // IE

var xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");

}

xmlHttp.open('POST', strURL, true);

xmlHttp.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');

xmlHttp.onreadystatechange = function() {

if (xmlHttp.readyState == 4) {

updatepage(xmlHttp.responseText);

}

}

xmlHttp.send(strURL);

}


function  updatepage(responsetext){

document.getElementById("date").innerHTML =
```

```
"<font color='blue' size='4'>" + responsetext + "</font>";

}

function showCurrentTime(){

var rnd = Math.random();

var url="time.php?id="+rnd;

postRequest(url);

}

</script>

</head>

<body>
<center>
<table border="0" cellpadding="0" cellspacing="1" width="464" bgcolor="#ccffcc">
<tr>
<td width="525" colspan="2" ><font color="blue">
<p align="center"><b>Show current time example in Ajax</b></font></td>
</tr>
<tr>
<td width="289" ><font color="blue"> Click to see Current Date and Time!</font></td>
<td width="236">
<form name="f1">

<p align="center">
 <input value="Show" type="button" onclick='JavaScript:showCurrentTime()'
name="Show">

</p>
<tr><td>
<div id="date"></div>

</td></tr>
</form>

</tr>

</table>
</center>


</body>

</html>
```

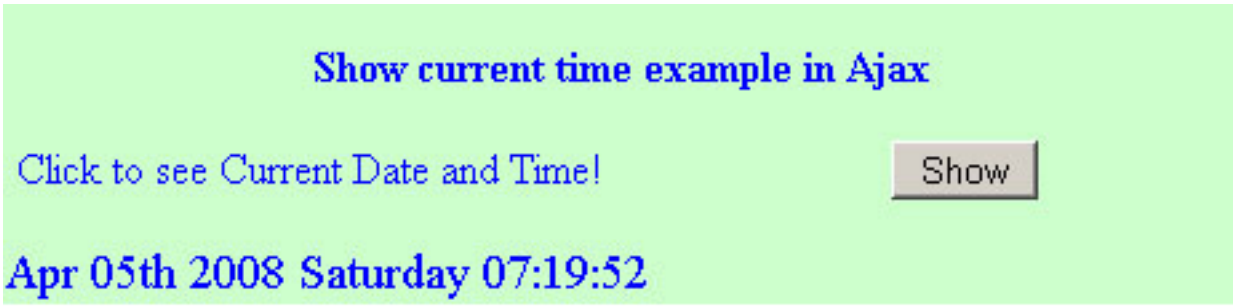
Ajax Examples

```
time.php
<?
print date("M dS Y l H:i:s");
?>
```

JavaScript code first tries to get HTTP request object maintaining the browser compatibility. In this example, this object is stored in the variable named “xmlHttp”. The request for "time.php" is sent to the server which calculates the current date and time. Now JavaScript method which is set to "onreadystatechange" property is called on getting the response back from the server. This method checks the status of the response and get the server response from responseText property of the xmlHttp object. Now the html div component of "date" id is populated with the server response text. Now date and time can be seen on the page.Output:



Click on Show button to get time!



Ajax Frameworks Types

Ajax Frameworks Types

Client Side vs. Server Side Frameworks

There are two types of Ajax based frameworks used in the web programming nowadays: Server-side Framework and Client-side Framework. Server-side framework is installed inside the server while Client-side framework entertains the user's browser to access the web. Similarly, the code executed on the Web server is considered as Server-side code while if executed on the user's browser is known as Client-side Framework.

Internet is an extremely wide area and because of this both the server-side and client –side programmers may not be in the contact and performs their work separately. Both the server-side and client-side functions differently and the code used in the frameworks also varies. The code of both the frameworks has its own pros and cons that suite to different sorts of developers.

Now the question arises, what is the Server-side framework and what is Client-side? In this article, you will get the answer of both these queries. You will also learn about what are the advantages and disadvantages of both these frameworks.

Server Side Framework

The framework that is installed inside the server is known as Server-side framework. The coding of server-side framework is actually not created in JavaScript but it is the same language that API provided for server-side framework. So, it is not mandatory for the developers to be skilled in JavaScript for server-side framework programming. Nearby all the frameworks in server-side accepts the coding of developers and automatically translates it into Ajax when user accesses the website.

Server-side framework can also be used for installing the specific functions from the library to translate and accessed effectively. The server-side framework exactly provides the desired functions and the library needed for developer to render the specific buttons and information in Ajax based website. The developers have to only customize the code during creating the program. Moreover, the server-side framework responds better because it handles a large number of user communications without intervention of server.

Google Web Toolkit (GWT) might be an excellent example of server-side framework that is used as a normal Java Swing kind of programming on the server side in web pages using the GWT API. The other server side frameworks are: Yahoo, Toolkit and DWR.

Server-side framework is basically used for server server-side programming that covers password protection, browser customization, form processing and building and displaying pages created from database.

Pros and Cons of Server Side Frameworks

There is no need for the developers to learn the extra language as server-side coding can be written in the existing language in which the server-side programmer is efficient.

Client Side Framework

As opposing the server-side, Client-side framework is accessed and executes within the user's browsers. For creating Ajax based client-side framework, developers must be efficient enough in

Ajax Frameworks Types

JavaScript as well as XML. Because Ajax-based client side website is fully depended upon Ajax and it also needs to a wide range of coding for different browsers.

The different uses of framework for different browsers make the Ajax client-side programming very complex. The frameworks written for Mozilla cannot be executed in Internet Explorer if XMLHttpRequest has not been defined and implemented for the Internet explorer too.

The Request XMLHttpRequest performs like a bridge between the server and the client. The developers have to generate different coding for different platforms to perform it successfully. That's why the developers must have the knowledge about which browser has been used by his client; otherwise the developer has to develop several customize client-side framework for running their own platforms.

Through Client-side framework, users can execute any technology that supports user's browser. In Client-side framework, users can modify several features besides codes without sending information to web server.

In the Client-side framework, user can handle multiple asynchronous request, can detect easily error handling and exceptions and can mix and match widget facilities to get extra flexibility in the programming. (Widgets are small interfaces or components that can be integrated with an application easily through client side framework)

Client-side framework is used in online games, customizing the display without reloading the page and getting data about the user's screen or browser. The examples of client-side framework are Prototype, DOJO Toolkit, Rico, Script.aculo.us and many more.

-

Pros and Cons of Client Side Frameworks

Client-side framework is more flexible than server-side due to multi-handing frameworks, easily error handling and exceptions while the adding of widgets makes it extra-flexible.

Ajax and XML: Five cool Ajax widgets

First published by IBM developerWorks at http://www.ibm.com/developerWorks. Visit ibm.com/developerWorks for more tutorials on open standard technologies, IBM products, and more.

Ajax and XML: Five cool Ajax widgets
Use Ajax and XML with new graphic tools to enhance your site

Level: Intermediate

Jack D Herrington (jherr@pobox.com), Senior Software Engineer, Leverage Software Inc.

16 Jan 2007
With the Web 2.0 wave came a whole new emphasis on the user experience. Part of that experience is the development novel ways to interact with and present information to users. Often, these new interfaces are called widgets and use Asynchronous JavaScript + XML (Ajax) to communicate with the server. Discover five widgets that you can use to enhance the interactivity of your site.

The Web 2.0 revolution emphasizes unique and novel ways to interact with customers on your Web site. A lot of these new, innovative techniques revolve around using graphics and widgets that communicate with the server to retrieve data for display. In this article, I introduce you to five such widgets — some open source, some licensed – that communicate with the server through Ajax and XML:

- **carousel:** This widget is a rolling image viewer that customers can use to scroll through a list of items, each portrayed by a small graphic. What you do when a user clicks an item is up to you. Examples of carousels in the wild include the Flikr site and the iTunes interface from Apple. This carousel is available at no cost and is based on the popular jQuery JavaScript framework.

- **SWF/Charts:** This Adobe Flash-based control reads XML located on the server for its charting data and styling options, then displays a chart based on the data. The interface is elegant, and the XML data is so easy to create that it's a snap to add dynamic graphing to your page.

- **SWF/Gauge:** A cousin to SWF/Charts, this Flash widget uses XML located on the server to build a completely customizable gauge display. The gauge can look like something from an airplane or a car or something more trendy. The choice is up to you.

- **In-place editing:** While not strictly a widget, an in-place editing control is an intuitive, interactive, and lightweight way to get information from users when they have it. In-place editing functionality comes with the Scriptaculous framework, which sits on top of the prototype.js library.

- **DHTML windows:** The DHTML window provides a mechanism for putting a modeless floating window on top of your page content. Users can move the window around, resize it, or dismiss it. The content of the window can either be specified by JavaScript code on the page or read

through Ajax from the server. This type of window is ideal for use as an alert mechanism or for bringing up small forms that don't deserve an entire page reload.

I start the show off with the SWF/Charts widget, because I think it's one of the easiest widgets to deploy. It also provides the biggest return for the effort.

The SWF/Charts widget

It's hard to argue with the old saying, "A picture is worth a thousand words," particularly where graphs are concerned. But graphing on the Web has always been a problem. Most Web frameworks lack a graphing tool right out of the box, although some include the graphics primitives for building images. This lack of functionality leaves you stuck building graphs on your own.

Wouldn't it be great if there were a widget that would just graph XML-encoded data? Turns out, there is one: SWF/Charts. To start using this widget, I download the SWF file from the site along with the extra SWF files that the widget uses. Then, I installed the files on my site and added a link to the SWF widget on the HTML page, as shown in Listing 1.

Listing 1. Chart_page.html

```
<html><body>

<object
  classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
 codebase="http://download.macromedia.com/pub.../swflash.cab#version=6,0,0,0"
 width="400" height="250">
<param name="movie"
 value="charts.swf?xml_source=chart_data.xml&library_path=charts_library">

<embed
 src="charts.swf?xml_source=chart_data.xml&library_path=charts_library"
 width="400" height="250"
 type="application/x-shockwave-flash"
 pluginspace="http://www.macromedia.com/go/getflashplayer">
</embed>
</object>

</body></html>
```

Charts.swf takes two parameters: the location of its libraries directory and the URL of the XML data. The XML data format is ridiculously easy. Listing 2 shows a simple example.

Listing 2. Chart_data.xml

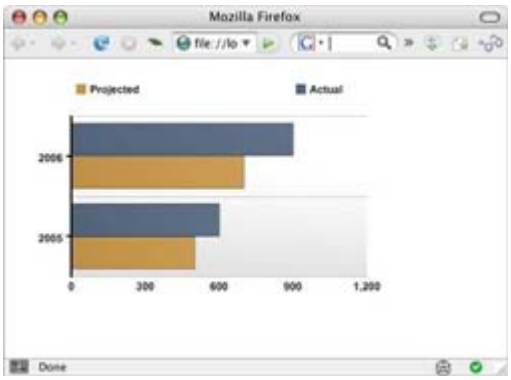
```
<chart>
 <chart_type>bar</chart_type>
 <chart_data>
  <row>
   <null/>
   <string>2005</string>
```

```
    <string>2006</string>
   </row>
   <row>
    <string>Projected</string>
    <number>500</number>
    <number>700</number>
   </row>
   <row>
    <string>Actual</string>
    <number>600</number>
    <number>900</number>
   </row>
  </chart_data>
</chart>
```

This file is primarily the data for the chart, along with some optional visual information. In this case, I'm specifying the chart type as a bar chart. The site from which I downloaded the SWF file has a lot more on the options you can set and the types of graphs that are available.

When I browse to the file in my Firefox browser, I see the graph shown in Figure 1.

Figure 1. The Chart Widget in action



As you can see, the default color scheme and the look of the chart is really slick. And the graph does the right thing of rounding up the axis values nicely. The overall effect is great with little effort on my part.

Obviously, you could replace the graph_data.xml file with a dynamic Web page: As long as the returned data is in the correct format, the graph control cares less. This is the case with all the examples in this article. In fact, you can run all the examples in a Web browser on local files without using a Web server (such as Apache Tomcat or IBM® WebSphere® Application Server) or Web programming language (for example, PHP, Microsoft® ASP.NET, Java™ 2 Enterprise Edition [Java EE]).

The SWF/Gauge widget

Another attractive way to present data is as a gauge. Personally, I'm not much on the gauge idea, because it takes up a lot of space to present just a little information. But gauges are a key feature of executive dashboards, so the ability to create them quickly is convenient.

But if the Web doesn't do simple bar charts very well, it certainly doesn't do circular gauges well. So, I went back to the same company that created XML/Graph, and wouldn't you know it? They also have a solution for gauges: XML/Gauge.

I'll start with the HTML page that embeds the SWF/Gauge widget, as shown in Listing 3.

Listing 3. Gauge_page.html

```
<html><body>

<object  classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
 codebase="http://download.macromedia.com/.../swflash.cab#version=6,0,0,0"
 width="110" height="55">
<param name=movie VALUE="gauge.swf?xml_source=gauge_data.xml">
<embed src="gauge.swf?xml_source=gauge_data.xml"
 width="110" height="55" type="application/x-shockwave-flash"
 pluginspace="http://www.macromedia.com/go/getflashplayer">

</embed></object>

</body></html>
```

That gauge.swf movie takes a single argument: the location of the data. In this case, the location is gauge_data.xml, which is shown in Listing 4_.

Listing 4. Gauge_data.xml

```
<gauge>

<circle fill_color="888888" start="275" fill_alpha="100"
 line_color="555555" line_thickness="3" line_alpha="90"
 radius="50" x="55" end="445" y="55"/>
<circle fill_color="99bbff" start="280" fill_alpha="90"
 line_thickness="4" line_alpha="20" radius="45" x="55"
 end="440" y="55"/>
<circle fill_color="666666" start="317" fill_alpha="100"
 line_color="333333" line_alpha="0" radius="44" x="55"
 end="322" y="55"/>
<circle fill_color="666666" start="337" fill_alpha="100"
 line_color="333333" line_alpha="0" radius="44" x="55"
 end="342" y="55"/>
<circle fill_color="666666" start="357" fill_alpha="100"
 line_color="333333" line_alpha="0" radius="44" x="55"
 end="362" y="55"/>
<circle fill_color="666666" start="377" fill_alpha="100"
```

```
 line_color=”333333" line_alpha=”0" radius=”44" x=”55"
 end=”382" y=”55"/>
<circle fill_color=”666666" start=”397" fill_alpha=”100"
 line_color=”333333" line_alpha=”0" radius=”44" x=”55"
 end=”402" y=”55"/>
<circle fill_color=”666666" start=”417" fill_alpha=”100"
 line_color=”333333" line_alpha=”0" radius=”44" x=”55"
 end=”422" y=”55"/>
<circle fill_color=”99bbff” start=”280" fill_alpha=”100"
 radius=”40" x=”55" end=”440" y=”55"/>
<circle fill_color=”FF4400" start=”280" fill_alpha=”100"
 radius=”44" x=”55" end=”310" y=”55"/>
<circle fill_color=”44FF00" start=”50" fill_alpha=”100"
 radius=”44" x=”55" end=”80" y=”55"/>
<circle fill_color=”99bbff” start=”280" fill_alpha=”80"
 radius=”40" x=”55" end=”440" y=”55"/>
<circle fill_color=”333333" start=”270" fill_alpha=”100"
 line_alpha=”0" radius=”20" x=”55" end=”450" y=”55"/>

<rotate start=”280" shake_span=”2" shadow_alpha=”15"
 step=”1" x=”55" span=”0" y=”55" shake_frequency=”20">
 <rect fill_color=”ffff00" fill_alpha=”90" line_alpha=”0"
   height=”40" x=”53" width=”4" y=”13"/>
</rotate>

<circle fill_color=”111111" start=”270" fill_alpha=”100"
 line_thickness=”5" line_alpha=”50" radius=”15" x=”55"
 end=”450" y=”55"/>

</gauge>
```

As you can see, SWF took a different approach with this widget. Instead of specifying the data for the gauge (or graph), I actually build the gauge out of graphics primitives such as circles, arcs, and rectangles.

Honestly, I prefer a set of canned gauges with which I can just supply the data. But this method works, and it allows me almost infinite room for tweaking — although I would have appreciated a few more canned examples that I could work from.

When I go to the page in my browser, I see the gauge shown in Figure 2.

Figure 2. The Gauge Widget in Action

You might think that with the specification of the graphics primitives, there isn't a lot to be gained on this widget. Not so. The primitives also include simple animation techniques so that you can bounce the needle around as well as sound and the ability to create hot-linking zones that navigate the browser when the user clicks them. In addition, you can think out of the box with this control, not just using it for gauges but using its simple graphics primitive language to build any type of image and simple animation.

In-place editing

Users now expect in-place editing from desktop applications, but this functionality is something rarely found on the Web, until now. With Web 2.0, interactivity becomes paramount, so techniques such as in-place editing are also more commonplace.

To implement in-place editing, you can either write it yourself or use one of the JavaScript frameworks to do the heavy lifting for you. One of the most popular toolkits is the Scriptaculous framework, which is built on top of the prototype.js library. The Scriptaculous library makes building an in-place edited control quite easy.

A simple HTML test file for in-place editing is shown in Listing 5.

Listing 5. Inplace.html

```
<html><head>
<script src="prototype.js"></script>
<script src="effects.js"></script>
<script src="controls.js"></script>
<script src="scriptaculous.js"></script>
</head><body>
<table width="100%">
<tr><th width="10%">Name</th>
<td width="90%"><p id="name">Candy bar</p></td>
</tr></table>
<script>
new Ajax.InPlaceEditor('name', 'submitted.html' );
</script>
</body>
</html>
```

To start, Inplace.html includes all the necessary JavaScript source files. Then, I put together a imple table with a paragraph containing the in-place editable data. At the end of the file, I insert a small bit of script that creates an InPlaceEditor object for the paragraph. That InPlaceEditor constructor takes as arguments the ID of the paragraph as well as the URL of the page that will handle the submission after I'm done editing. In this case, that page is submitted.html; but in reality, it would be an ASP.NET, Java EE, or PHP page or some other dynamic Web technology.

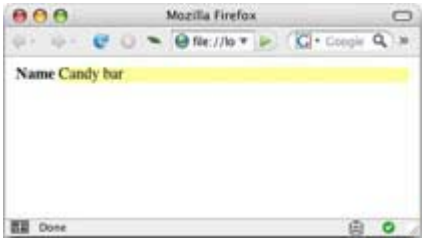
Listing 6 shows the simple submitted.html file.

Listing 6. Submitted.html

```
<p>Name changed!</p>
```

Now to test it. I first open my browser to the HTML file. There, I see the original text. As I mouse over the text, it turns yellow, as shown in Figure 3.

Figure 3. The starting point of in-place editing



This yellow highlighting is a visual indicator to users that they can edit the field by clicking it. So, I click the field and get the **Name** field, an **ok** button, and a **cancel** link, as shown in Figure 4.

Figure 4. Editing the text after clicking it

I then change the text and click **ok**, which posts the data to the server (or the submitted.html page, in this case). The server then returns the HTML page that should replace the original text. In this case, I send back Name changed! (as shown in Figure 5); in reality, it would probably be the new value of the data.

Figure 5. The new content after clicking ok

Simple interface upgrades like these can make a world of difference in the usability of your application. Waiting for page loads — especially from slow servers — gives the impression of a clunky, old-style interface. Using simple tools such as this in-place editor can really spruce up your application with very little in terms of implementation complexity.

DHTML windows

It's probably a good thing that browsers make it difficult to build modal windows into Web pages. But sometimes, small windows can be a good thing. They are handy to display alerts or to bring up small forms. They're also a great way to launch annoying ads that cover the content of the page. Oh, wait: Scrap that last one.

Anyway, as I said, it's not easy to build windows for Dynamic HTML (DHTML) pages. So I was happy when I found this extremely robust window package based on the popular Protoype.js library. Not only was it easy to use, but the interface was skinnable and works well on every browser. Listing 7 shows the window.html page.

Listing 7. Window.html

```
<html>
<head>
<link href="default.css" rel="stylesheet" type="text/css" />
<script src="prototype.js"></script>
<script src="window.js"></script>
</head>
<body>
<script>
var win = new Window( 'myPopup', {
  title: "Terms and Conditions",
  top:70, left:100, width:300, height:200,
  resizable: true, url: "terms.html",
  showEffectOptions: { duration: 3 }
 }
);
win.show();
</script>
</body>
</html>
```

I first bring the prototype.js and window.js source files into the header. Then, I build the pop object with the parameters I like, including the size, the location, the title, and the URL of the page from which the widget should get its content. Loading the content from a page through Ajax is just one way of getting the contents, though; you can also set them dynamically through JavaScript code or wrap the window around an existing tag on the page.
In this case, I reference the terms.html file shown in Listing 8.

Listing 8. Terms.html

```
<html><body bgcolor=”white”>
<h1>Terms and Conditions</h1>
<p>In order to use this site you must comply
with the following conditions...</p>
</body></html>
```

When I launch the page in my browser, I see the window shown in Figure 6.

Figure 6. The initial window

No, that’s not just two Mac windows on top of each other. That’s a Mac-looking fake DHTML window inside a real Firefox browser window. But it looks and feels the same anyway.
I can stretch and move the window around, as shown in Figure 7.

Figure 7. The window after moving and resizing it

I looked at several DHTML window libraries, both for this article and for my own work, and I can tell you with some confidence that this one has the best feel to me. Other window packages had rendering problems, rendered in segments, or behaved badly when I resized them. This one feels very much like a real window that’s just trapped inside the browser.

The carousel widget

Anyone who has done a significant amount of user interface (UI) work can tell you that screen real estate is critical. It's important to squeeze as much data as you can into a given space without it feeling compressed. So, I was pretty impressed when I first saw a carousel control in Apple iTunes. A *carousel control* shows several images in a fixed block of space. To the left and right of the block of images are left and right arrows. If you click the arrows, the images shift to the left or right and are replaced with a new set of images. In iTunes, the images were album covers, and there was a carousel control for each genre.

The space savings are significant: You can put 30 album covers in the space of three and still show each at a reasonable size. And the control is intuitive. It's like a simplified scrollbar.

The downside is that carousels aren't easy to implement, especially because part of the allure is the animation of the images moving to the left or the right. So I was happy to see an open source carousel called *carousel* built on the jQuery JavaScript framework.

I implemented a simple carousel widget on the Web page shown in Listing 9.

Listing 9. Carousel.html

```
<html>
<head>
<script type="text/javascript" src="js/jquery-1.0.3.js"></script>
<script type="text/javascript" src="js/jcarousel.js"></script>
<style type="text/css">
#mycarousel { display: none; }
.jcarousel-scope {  position: relative; width: 255px;
  -moz-border-radius: 10px; background: #D4D0C8;
  border: 1px solid #808080; padding: 20px 45px; }
.jcarousel-list li { width: 81px; height: 81px;
  margin-right: 7px; }
.jcarousel-list li img { border: 1px solid #808080; }
.jcarousel-list li a { display:block;  outline: none;
  border: 2px solid #D4D0C8; -moz-outline:none; }
.jcarousel-list li a:hover { border: 2px solid #808080; }
.jcarousel-next {  position: absolute; top: 45px;
  right: 5px; cursor: pointer; }
.jcarousel-next-disabled { cursor: default; }
.jcarousel-prev { position: absolute; top: 45px;
  left: 5px; cursor: pointer; }
.jcarousel-prev-disabled { cursor: default; }
.loading { position: absolute; top: 0px;
  right: 0px; display: none; }
</style>
<script type="text/javascript">
function loadItemHandler( carousel, start, last, available ) {
  if (available) { carousel.loaded(); return; }
  var cr = carousel;
  jQuery.get("data.xml", function(data) { appendItemCallback(cr, start, last, data); });
};

function appendItemCallback( carousel, start, last, data ) {
```

```
  var items = data.match( /(\<img .*?\>)/g );

  for (i = start; i <= last; i++) {
    if ( items[ i - 1 ] == undefined ) break;
    var item = carousel.add( i, getItemHTML( items[i-1]) );
    item.each(function() {
      jQuery("a.thickbox", this).click(function() {
        var t = this.title || this.name || null;
        var g = this.rel || false;
         TB_show(t,this.href,g);
        this.blur();
        return false;
      });
    });
  }
  carousel.loaded();
};

function getItemHTML( item ) {
  var found = item.match( /href=\"(.*?)\"/ );
  var url   = jQuery.trim(found[1]);
  var title = jQuery.trim(found[1]);
  var url_m = url.replace(/_s.jpg/g, '_m.jpg');
  return '<a href="' + url_m +
  '" title="' + title +
  '" class="thickbox"><img src="' + url +
  '" width="' + 75 + '" height="' + 75 +
  '" alt="' + title + '" /></a>';
};

var nextOver = function() {
  jQuery(this).attr("src", "img/horizontal-ie7/next-over.gif"); };

var nextOut = function() {
  jQuery(this).attr("src", "img/horizontal-ie7/next.gif"); };

var nextDown = function() {
  jQuery(this).attr("src", "img/horizontal-ie7/next-down.gif"); };

function nextButtonStateHandler(carousel, button, enabling) {
  if (enabling) {
    jQuery(button).attr("src", "img/horizontal-ie7/next.gif")
        .mouseover(nextOver).mouseout(nextOut).mousedown(nextDown);
  } else {
    jQuery(button).attr("src", "img/horizontal-ie7/next-disabled.gif")
        .unmouseover(nextOver).unmouseout(nextOut).unmousedown(nextDown);
  }
}

var prevOver = function() {
  jQuery(this).attr("src", "img/horizontal-ie7/prev-over.gif"); };
```

```
var prevOut = function() {
 jQuery(this).attr("src", "img/horizontal-ie7/prev.gif"); };

var prevDown = function() {
 jQuery(this).attr("src", "img/horizontal-ie7/prev-down.gif"); };

function prevButtonStateHandler(carousel, button, enabling) {
 if (enabling) {
   jQuery(button).attr("src", "img/horizontal-ie7/prev.gif")
     .mouseover(prevOver).mouseout(prevOut).mousedown(prevDown);
 } else {
   jQuery(button).attr("src", "img/horizontal-ie7/prev-disabled.gif")
     .unmouseover(prevOver).unmouseout(prevOut).unmousedown(prevDown);
 }
}

jQuery(document).ready(function() {
 jQuery().ajaxStart(function() { jQuery(".loading").show(); });
 jQuery().ajaxStop(function() { jQuery(".loading").hide(); });
 jQuery("#mycarousel").jcarousel({
   itemVisible: 3, itemScroll: 2, wrap: true,
  loadItemHandler: loadItemHandler,
    nextButtonStateHandler: nextButtonStateHandler,
   prevButtonStateHandler: prevButtonStateHandler
 });
});
</script></head><body><div id="mycarousel">
<div class="loading">
<img src="img/loading.gif" width="16" height="16" border="0" />Loading...</div>
<img src="img/horizontal-ie7/prev.gif" border="0" class="jcarousel-prev" />
<img src="img/horizontal-ie7/next.gif" border="0" class="jcarousel-next" />
<ul></ul>
</div></body></html>
```

Yes, there is a lot more to it than in the previous examples. But most of the code is setting up the graphics and interpreting the Ajax data returned from the server. In fact, most of the code for this article is based on one of the examples provided with the download. So, I didn't have to learn much or read any documentation to use the control.
The data for the carousel is shown in Listing 10.

Listing 10. Data.xml

```
<images>
<img href="pics/image1.jpg" />
<img href="pics/image2.jpg" />
<img href="pics/image3.jpg" />
<img href="pics/image4.jpg" />
</images>
```

In this case, the file is just in a simple XML format that has an <images> tag with a set of <img> ags in it that hold the URL for each image. You can use whatever format you like, because the ontrol is not natively an Ajax widget. I'm writing the code that interprets the XML and creates each slide element in the carousel. The end result is shown in Figure 8.

Figure 8. The image carousel on the page

I can click the image and go to the page with the image (or to any URL I specify). Or, I can click the right or left arrow to scroll around the carousel to see more images. The effect is really quite impressive.

Conclusion

I showed you just a handful of the widgets and tools available on the Web both commercially and at no cost. Many of the tools I looked at while I researched this article didn't use Ajax and as such didn't fit into the topic. However, they were noteworthy on their own. In particular, I was impressed by the number of high-quality, open source WYSIWYG editors available for download. I often have customers frustrated when they have to use HTML in text boxes to get content onto their sites with bold, italics, links, images, and so on. These editors hide all the HTML and give users an editing feel that's similar to a word processing application.

In addition to the WYSIWYG editors, you can find solutions for progress bars, tabbed dialog boxes, accordion controls, clocks, date pickers, RSS and Outline Processor Markup Language (OPML) readers — even interactive terminal windows. Certainly, before you build your own DHTML or Flash controls, you should look at what's available (often at no cost) on the Internet. With widgets such as these, you can add a lot of interactivity to your site without a lot of effort.

Five common Ajax patterns

First published by IBM developerWorks at http://www.ibm.com/developerWorks. Visit ibm.com/developerWorks for more tutorials on open standard technologies, IBM products, and more.

Ajax and XML: Five common Ajax patterns
Helpful Ajax design patterns you can use today

Level: Intermediate

Jack D Herrington (jherr@pobox.com), Senior Software Engineer, Leverage Software Inc.

06 Mar 2007

Asynchronous JavaScript + XML (Ajax) was certainly the technology buzzword of 2006 and looks to do just as well or better in 2007. But what does it really mean for your application? And which common architectural patterns are used widely in Ajax applications? Discover five common Ajax design patterns that you can use as a basis for your own work.

Sure, Ajax is the Web 2.0 buzzword that everyone wants associated with their site. But what does it really mean? And how are engineers integrating it into their sites at an architectural level. In this article, I cover the basics of Ajax and show some Ajax design patterns that have become proven best practice with Web 2.0 development.

To start, *Ajax* is just a buzzword that covers a set of technologies, including Dynamic HTML (DHTML) and the XMLHTTPRequest object. DHTML is a combination of three elements: Hypertext Markup Language (HTML), JavaScript code, and Cascading Style Sheets (CSS). Using JavaScript code on a Web page, you can change the page dynamically to add, remove, or change the content. That's the *dynamic* portion of DHTML. JavaScript code uses the XMLHTTPRequest object to request data from the server after the page has been loaded.

The combination of these two elements — requesting data from the server on the fly and changing the page to use the data — is the essence of what's called *Ajax* and the dynamic nature of Web 2.0 sites.

But that doesn't really tell you how these features are used in the real world and how you should use them on your site. For that, you need a set of simple *design patterns.* If you are unfamiliar with that term, it comes from the excellent book of the same name (see Resources). That book provided a set of implementation patterns for the common tasks that confront engineers. The book provided not just best practices for how to design systems but also a terminology that engineers can use to talk about their code.

This article presents five common Ajax design patterns. They vary in using HTML, XML, and JavaScript code to get data from the server. I start with the simplest pattern, which is to update your page with new HTML from the server.

Pattern 1. Replacing HTML segments

Perhaps the most common Ajax task is to request updated HTML from the server and update a

portion of the page with it. You can do this can periodically — for example, to update stock quotes. Or you can update on demand — for example, in response to a search request.
The code in Listing 1 requests a page from the server, and then places that content into a tag in the body of the page.

Listing 1. Pat1_replace_div.html

```
<html>
<script>
var req = null;
function processReqChange() {
  if (req.readyState == 4 && req.status == 200 ) {
    var dobj = document.getElementById( 'htmlDiv' );
    dobj.innerHTML = req.responseText;
  }
}

function loadUrl( url ) {
  if(window.XMLHttpRequest) {
    try { req = new XMLHttpRequest();
    } catch(e) { req = false; }
  } else if(window.ActiveXObject) {
    try { req = new ActiveXObject('Msxml2.XMLHTTP');
    } catch(e) {
    try { req = new ActiveXObject('Microsoft.XMLHTTP');
    } catch(e) { req = false; }
  } }
  if(req) {
    req.onreadystatechange = processReqChange;
    req.open('GET', url, true);
    req.send('');
  }
}

var url = window.location.toString();
url = url.replace( /pat1_replace_div.html/, 'pat1_content.html' );
loadUrl( url );
</script>
<body>
Dynamic content is shown between here:<br/>
<div id="htmlDiv" style="border:1px solid black;padding:10px;">
</div>
And here.<br/>
</body>
</html>
```

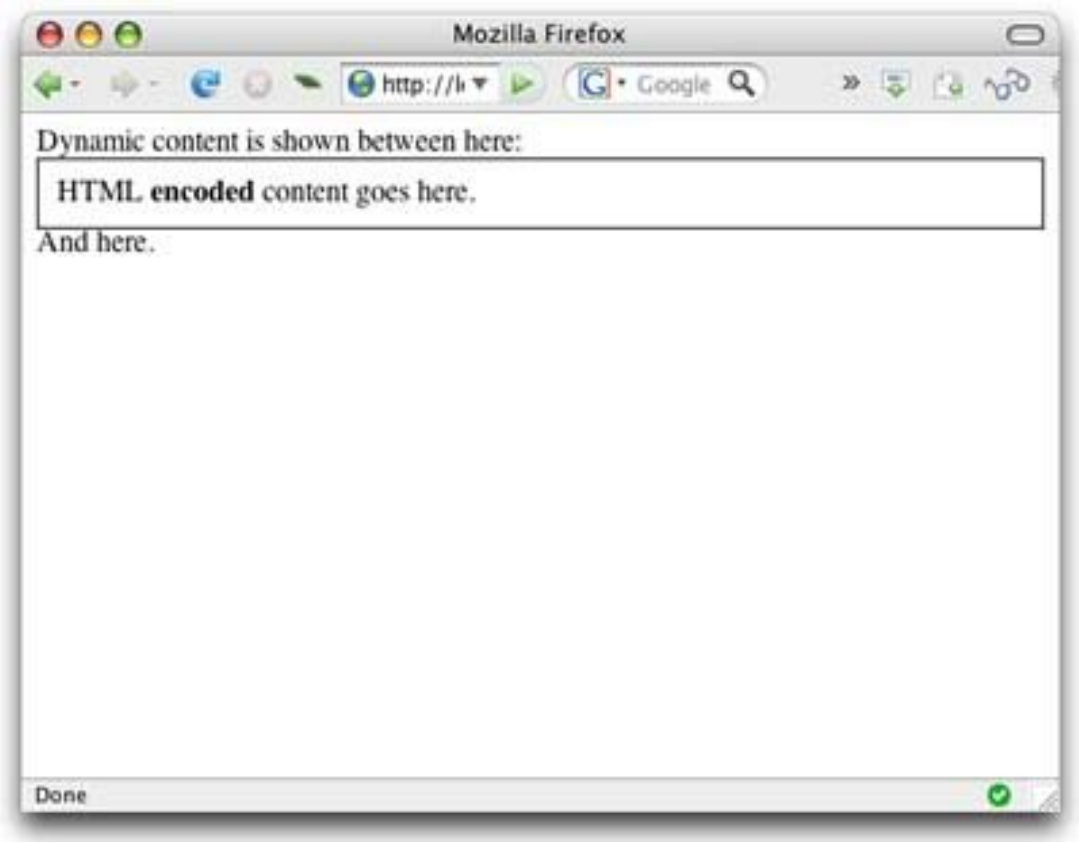
Listing 2 shows the content that the code is requesting.

Listing 2. Pat1_content.html

HTML encoded content goes here.

When I load the page in Firefox, I see the result shown in Figure 1.

Figure 1. The page with the replaced tag



Go back to the code in Listing 1 and look at a few things. The first thing to notice is the loadUrl() function, which requests a URL from the server. This function uses the XMLHTTPRequest object to ask the server for the new content. It also specifies a callback function — in this case, processReqChange — that's called when the browser has received the content.
The processReqChange function then inspects the object to see whether the request has been completed. If it has, the function sets the innerHTML of the tag in the page into the text of the response.

The use of the tag as a placeholder for dynamic content is a staple of Ajax code. These tags have no visible presence (unless you add borders and such, as I have), but they act as a good marker for where content should go. Engineers also use the tag for replaceable segments, as I demonstrate later. The difference between a and a tag is that the former imposes a line break (like a paragraph), while the latter delineates a section of inline text.
Getting back to the processReqChange function for a moment, it's important that the function

check the value of both the status and the readyState value. While some browsers will call the function only when the request is complete, other browsers will call back continuously to tell the code that the request is still running.

The tabbed display variant

Another variant of this pattern is to create a tabbed style of display. Listing 3 shows a simple tabbed Ajax interface.

Listing 3. Pat1_tabs.html

```
<html>
<script>
var req = null;
function processReqChange() {
  if (req.readyState == 4 && req.status == 200 ) {
    var dobj = document.getElementById( 'tabDiv' );
    dobj.innerHTML = req.responseText;
  }
}

function loadUrl( tab ) {
  var url = window.location.toString();
  url = url.replace( /pat1_tabs.html/, tab );
  ...
}

function tab1() { loadUrl( 'pat1_tab1_content.html' ); }
function tab2() { loadUrl( 'pat1_tab2_content.html' ); }
tab1();
</script>
<body>
<a href="javascript: void tab1();">Tab 1<a>
<a href="javascript: void tab2();">Tab 2<a>
<div id="tabDiv" style="border:1px solid black;padding:10px;">
</div>
</body>
</html>
```

Listing 4 shows the content for the first tab.

Listing 4. Pat1_tab1_content.html

```
Tab 1 content
```

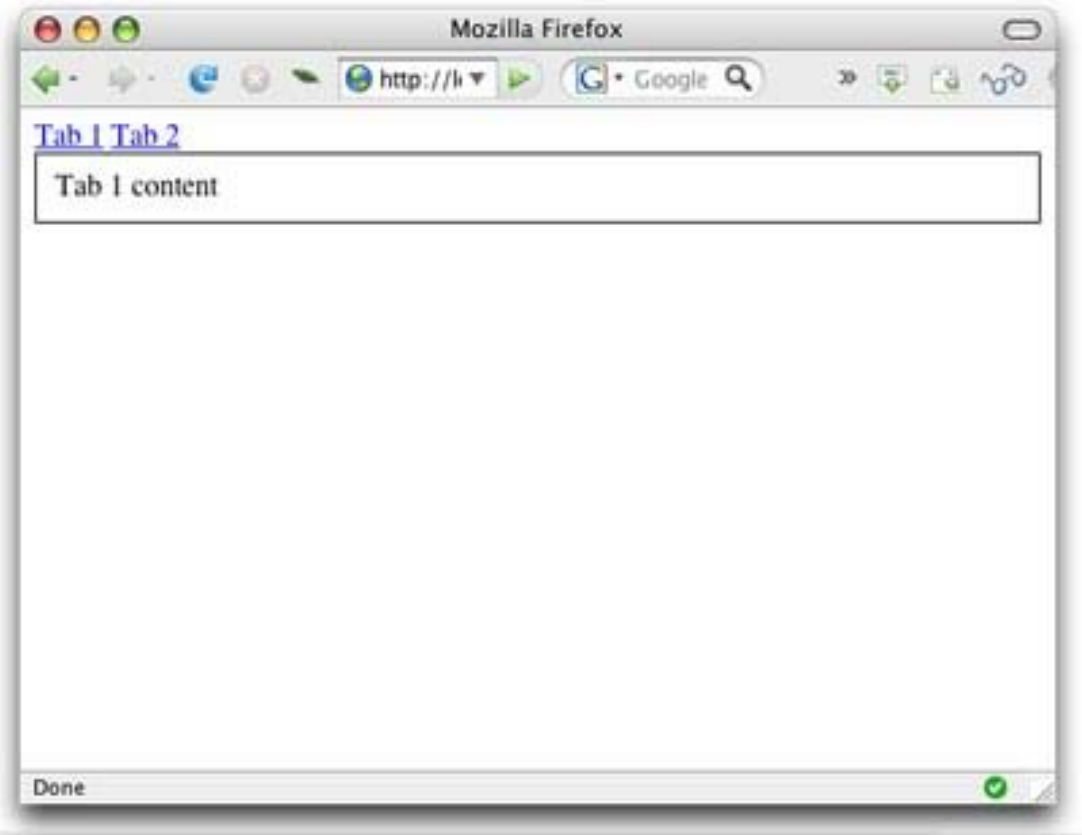
And Listing 5 shows the content for the second tab.

Listing 5. Pat1_tab2_content.html

```
Tab 2 content
```

When I bring this page up in my browser, I see the first tab, as shown in Figure 2.

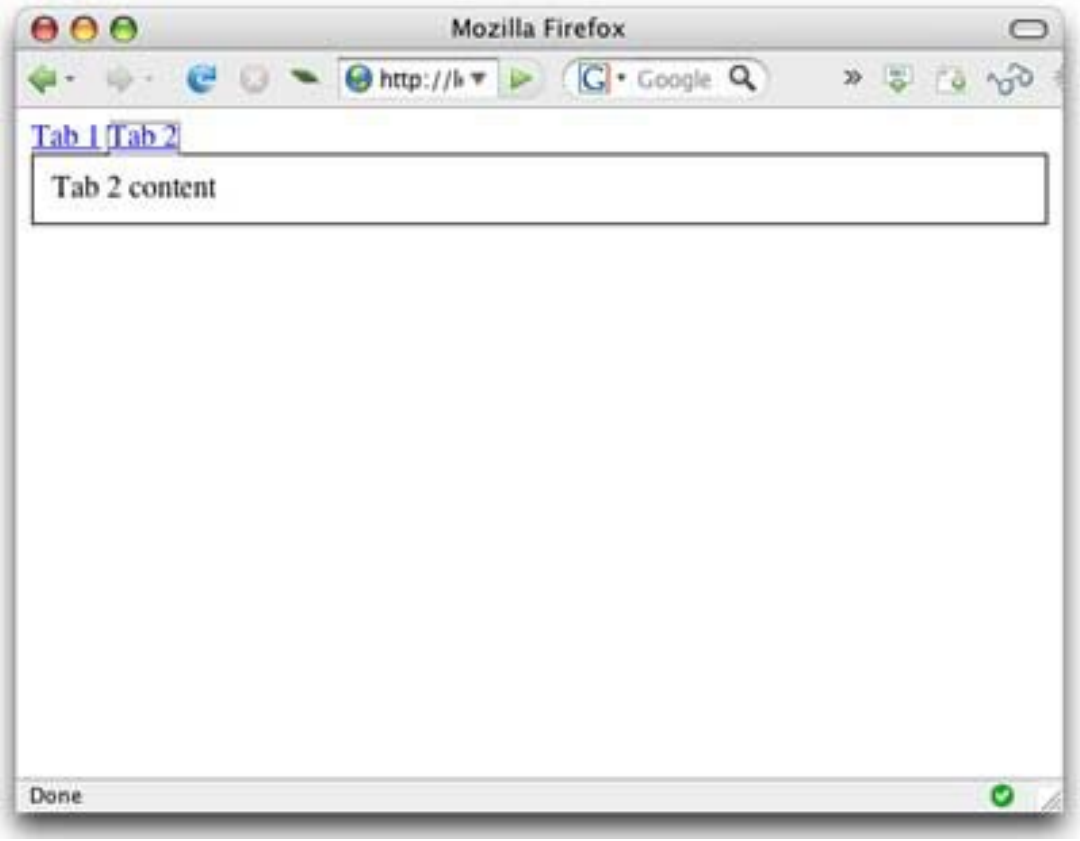
Figure 2. The content for the first tab



I then click the link for the second tab. The browser retrieves the second tab's contents and shows

it in the tab area, as shown in Figure 3.

Figure 3. The content for the second tab



This is the quintessential use of this design pattern — to take requests from the user and update a portion of the display with the new material, in this case, creating the illusion of a tabbed display. The value from the application side is that you can downloaded a much lighter-weight page to customers, who can then access the material they want on demand.

Before Ajax, the common technique was to have both tabs on the page, then hide or show them on demand. This meant that the HTML for the second tab was created even if it was never viewed, wasting both server time and bandwidth. With this new Ajax method, the HTML for the second tab is created only when the user requests it.

The read more variant

Yet another variation on this theme is the **Read more** link, as shown in Figure 4.

Figure 4. The Read more link on my boring blog entry

Suppose that I really wanted to read more about the continuing adventures of my dog walk. I can click the **Read more** link and have that link replaced with the complete engrossing story, as shown

in Figure 5.

Figure 5. The page after clicking the Read more link

The value for customers is that they get more material seamlessly without a page refresh. Listing 6 shows the code for this page.

Listing 6. Pat1_readmore.html

```
<html>
<script>
var req = null;
function processReqChange() {
  if (req.readyState == 4 && req.status == 200 ) {
    var dobj = document.getElementById( "moreSpan" );
    dobj.innerHTML = req.responseText;
  }
}
```

```
function loadUrl( url ) { ... }

function getMore()
{
 var url = window.location.toString();
 url = url.replace( /pat1_readmore.html/, 'pat1_readmore_content.html' );
 loadUrl( url );
}
</script>
<body>
<h1>Walking the dog</h1>
I took my dog for a walk today.
<span id="moreSpan">
<a href="javascript: void getMore()">Read more...</a>
</span>
</body>
</html>
```

Listing 7 shows the content for the "read more" section.

Listing 7. Pat1_readmore_content.html

It was a nice day out. Warm and sunny. My dog liked getting out for a stretch.

This code demonstrates the use of the tag instead of the tag. The approach you use depends on the requirements of your user interface (UI). But as you can see, it's easy to use either approach.

Getting new HTML for the page is one thing, but what about when you want the JavaScript code on the page to actually do something intelligent with the data. How do you get the data to the browser in a structured way? Why XML, of course.

Pattern 2. Reading XML data

For some reason, Ajax has become synonymous with XML, even though XML isn't strictly required. As you can see from the examples above, you can return straight text or even fragments of HTML — or Extensible HTML (XHTML) — code. But sending XML can have its rewards.

Listing 8 shows Ajax code that requests records about books from the server, then displays that data in a table within the page.

Listing 8. Pat2_xml.html

```
<html>
<head>
<script>
var req = null;
function processReqChange() {
 if (req.readyState == 4 && req.status == 200 && req.responseXML ) {
   var dtable = document.getElementById( 'dataBody' );
   var nl = req.responseXML.getElementsByTagName( 'book' );
   for( var i = 0; i < nl.length; i++ ) {
```

```
      var nli = nl.item( i );
      var elAuthor = nli.getElementsByTagName( 'author' );
      var author = elAuthor.item(0).firstChild.nodeValue;
     var elTitle = nli.getElementsByTagName( 'title' );
     var title = elTitle.item(0).firstChild.nodeValue;

      var elTr = dtable.insertRow( -1 );

      var elAuthorTd = elTr.insertCell( -1 );
      elAuthorTd.innerHTML = author;

      var elTitleTd = elTr.insertCell( -1 );
     elTitleTd.innerHTML = title;
} } }

function loadXMLDoc( url ) {
  if(window.XMLHttpRequest) {
    try { req = new XMLHttpRequest();
    } catch(e) { req = false; }
  } else if(window.ActiveXObject) {
    try { req = new ActiveXObject('Msxml2.XMLHTTP');
    } catch(e) {
    try { req = new ActiveXObject('Microsoft.XMLHTTP');
    } catch(e) { req = false; }
  } }
  if(req) {
    req.onreadystatechange = processReqChange;
    req.open('GET', url, true);
    req.send('');
  }
}

var url = window.location.toString();
url = url.replace( /pat2_xml.html/, 'pat2_xml_data.xml' );
loadXMLDoc( url );
</script>
</head>
<body>
<table cellspacing="0" cellpadding="3" width="100%">
<tbody id="dataBody">
<tr>
  <th width="20%">Author</th>
  <th width="80%">Title</th>
</tr>
</tbody>
</table>
</body>
</html>
```

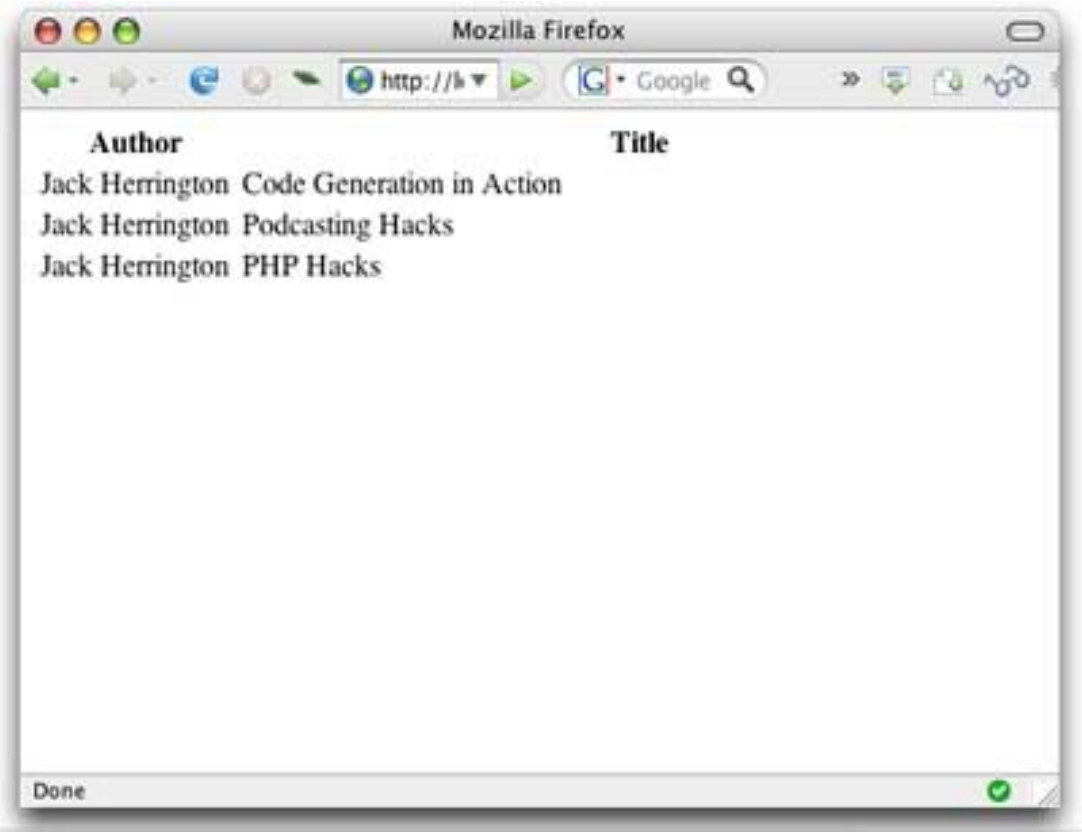
Listing 9 shows the data for the page.

Listing 9. Pat2_xml_data.xml

```
<books>
  <book>
    <author>Jack Herrington</author>
    <title>Code Generation in Action</title>
  </book>
  <book>
    <author>Jack Herrington</author>
    <title>Podcasting Hacks</title>
  </book>
  <book>
    <author>Jack Herrington</author>
    <title>PHP Hacks</title>
  </book>
</books>
```

When I load the page in my browser, I see the result shown in Figure 6.

Figure 6. The XML display page



The big difference between this page and the pages in the previous pattern is in the processReqChange function. Instead of looking at responseText, you now look at responseXML, an XML Document Object Model (DOM) available only if the response from the server was properly encoded XML. Using responseXML, I request the list of <book> tags from the XML document. I then get the <title> and <author> elements from each. Next, I add a row to the table for each book and cells to each row to contain the author and title data. This is a pretty rudimentary use of the XML data. More sophisticated JavaScript code can perform client-side sorting or searching based on the returned data. Unfortunately, the downside of transferring XML data is that it takes some time for the browser to parse through the XML document. Also, the JavaScript code to find the data in the XML can be complex (as seen in Listing 8). The alternative is to request JavaScript code from the server.

Pattern 3. Reading JavaScript data

Requesting JavaScript data from the server is a technique that often goes by the classy code name *JavaScript Object Notation* (JSON). The value of returning JavaScript data is that it's efficient for the browser to parse and creates JavaScript data structures, which are a lot easier to use. Let me revise the code in Listing 8 that read XML from the server to read JavaScript data from the server, instead. This new code is shown in Listing 10.

Listing 10. Pat3_js.html

```
<html><head><script>
var req = null;
function processReqChange() {
  if (req.readyState == 4 && req.status == 200 ) {
    var dtable = document.getElementById( 'dataBody' );
    var books = eval( req.responseText );
    for( var b in books ) {
      var elTr = dtable.insertRow( -1 );

      var elAuthorTd = elTr.insertCell( -1 );
      elAuthorTd.innerHTML = books[b].author;

     var elTitleTd = elTr.insertCell( -1 );
     elTitleTd.innerHTML = books[b].title;
} } }

...
```

All the HTML code remains the same. The processReqChange function simply changes to read an eval that the JavaScript data returned from the server. The function then uses the JavaScript objects that come out of the eval as the source of the data, which is then added to the table. Listing 11 shows the JavaScript data from the server.

Listing 11. Pat3_js_data.js

```
[ { author: 'Jack Herrington', title: 'Code Generation in Action' },
{ author: 'Jack Herrington', title: 'Podcasting Hacks' },
{ author: 'Jack Herrington', title: 'PHP Hacks' }
]
```

Five common Ajax patterns

It's easy to see why so many Ajax application engineers prefer to use JavaScript code instead of XML to encode the data. The JavaScript code is easier to read and manage as well as easier for the browser to process.

With all this data-gathering and display, you see that the key to Ajax is the display of current data — the important part there being *current.* So, how do you ensure that you're always getting fresh data from the server?

Pattern 4. Avoiding browser cache
Browsers attempt to optimize Web traffic, so if you ask for the same URL twice, it's likely that rather than request the page again, your browser will simply use the page stored in the browser cache. So, another common pattern in Ajax applications is the use of some randomizing element in the URL to ensure that the browser doesn't return a cached result.

My favorite technique is to add the numeric value of the current time to the URL. Listing 12 shows this technique.

Listing 12. Pat4_cache.html

```
<html>
<script>
...

function loadUrl( url ) {
  url = url + "?t="+((new Date()).valueOf());
  ...
}

...
```

This is the code from Listing 1 but with the addition of some JavaScript text manipulation of the URL string. I append to the URL a new parameter called t that has the value of the time. It doesn't really matter whether the server recognizes the value. It's just a way to ensure that the browser ignores its URL-based page cache.

Pattern 5. Replacing multiple HTML segments
The final pattern I demonstrate is an advanced version of the first pattern: the replacement of a tag with content from the server. A common problem in Web applications is that in response to user input, several areas of the display must be updated. For example, in a stock quote application, one part of the display might show the most recent quote, while another portion of the display shows a list of the most recent values.

To update multiple areas of the display, I use an XML response from the server that contains data for both sections. Then, I use a regular expression to break out the individual sections from the response. Listing 13 shows this technique.

Listing 13. Pat5_multi_segment.html

```
<html>
<head>
```

```
<script>
var req = null;
function processReqChange() {
  if (req.readyState == 4 && req.status == 200 ) {
    var one = req.responseText.match( /\<one\>(.*?)\<\/one\>/ );
    document.getElementById( 'divOne' ).innerHTML = one[1];
    var two = req.responseText.match( /\<two\>(.*?)\<\/two\>/ );
    document.getElementById( 'divTwo' ).innerHTML = two[1];
} }

function loadXMLDoc( url ) { ... }

var url = window.location.toString();
url = url.replace( /pat5_multi_segment.html/, 'pat5_data.xml' );
loadXMLDoc( url );
</script>
</head>
<body>

This is the content for segment one:<br/>
<div id="divOne" style="border:1px solid black;padding:10px;">
</div>
And segment two:<br/>
<div id="divTwo" style="border:1px solid black;padding:10px;">
</div>

</body>
</html>
```

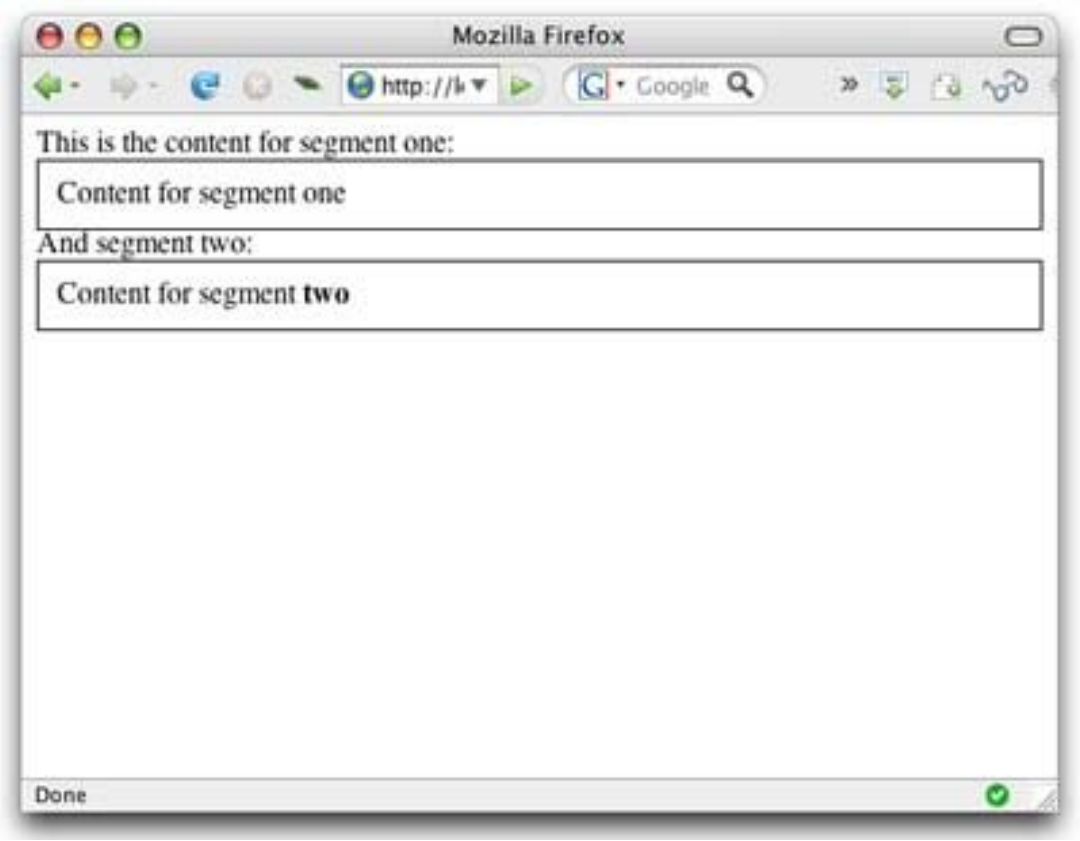
Listing 14 shows the data from the server.

Listing 14. Pat5_data.xml

```
<segments>
  <one>Content for segment one</one>
  <two>Content for segment <b>two</b></two>
</segments>
```

When I load this code in my browser, I see the result shown in Figure 7.

Figure 7. The two-segment display updated with data from the server



In the page code, I could have used the XML response, as the material returned from the server is valid XML. But it was easier to use regular expressions instead to crack the individual segments from the XML code.

Conclusion

Ajax is as powerful as it is misunderstood and misused. The patterns I've shown in this article provide a good jumping-off point for using Ajax in your Web application. But in addition to using the code provided here, I recommend having a look at some of the great Ajax and Web UI libraries that have come along with the Web 2.0 revolution. Chief among these is the Prototype.js library, which provides easy methods to get data to and from the server as well as cross-browser-compliant methods to update Web page content. The value of using these libraries is that dedicated engineers maintain and test them on a wide variety of browsers and platforms, which can save you a lot of work and headache.

Either way you cut it, Ajax as demonstrated by the patterns in this article is something you should check into to add dynamic behavior to your applications.

Advertise with JavaJazzUp

We are the top most providers of technology stuffs to the java community. Our technology portal network is providing standard tutorials, articles, news and reviews on the Java technologies to the industrial technocrats. Our network is getting around 3 million hits per month and its increasing with a great pace.

For a long time we have endeavored to provide quality information to our readers. Furthermore, we have succeeded in the dissemination of the information on technical and scientific facets of IT community providing an added value and returns to the readers.

We have serious folks that depend on our site for real solutions to development problems.

JavaJazzUp Network comprises of :

http://www.roseindia.net
http://www.newstrackindia.com
http://www.javajazzup.com
http://www.allcooljobs.com

Advertisement Options:

Banner	Size	Page Views	Monthly
Top Banner	470*80	5,00,000	USD 2,000
Box Banner	125 * 125	5,00,000	USD 800
Banner	460x60	5,00,000	USD 1,200
Pay Links		Un Limited	USD 1,000
Pop Up Banners		Un Limited	USD 4,000

The http://www.roseindia.net network is the "real deal" for technical Java professionals. Contact me today to discuss your customized sponsorship program. You may also ask about advertising on other Technology Network.

Deepak Kumar
deepak@roseindia.net

Valued JavaJazzup Readers Community

We invite you to post Java-technology oriented stuff. It would be our pleasure to give space to your posts in JavaJazzup.

Contribute to Readers Forum

If theres something youre curious about, were confident that your curiosity, combined with the knowledge of other participants, will be enough to generate a useful and exciting Readers Forum. If theres a topic you feel needs to be discussed at JavaJazzup, its up to you to get it discussed.

Convene a discussion on a specific subject

If you have a topic youd like to talk about . Whether its something you think lots of people will be interested in, or a narrow topic only a few people may care about, your article will attract people interested in talking about it at the Readers Forum. If you like, you can prepare a really a good article to explain what youre interested to tell java technocrates about.

Sharing Expertise on Java Technologies

If youre a great expert on a subject in java, the years you spent developing that expertise and want to share it with others. If theres something youre an expert on that you think other technocrates might like to know about, wed love to set you up in the Readers Forum and let people ask you questions.

Show your innovation

We invite people to demonstrate innovative ideas and projects. These can be online or technology-related innovations that would bring you a great appreciations and recognition among the java technocrates around the globe.

Hands-on technology demonstrations

Some people are Internet experts. Some are barely familiar with the web. If you'd like to show others aroud some familiar sites and tools, that would be great. It would be our pleasure to give you a chance to provide your demonstrations on such issues : How to set up a blog, how to get your images onto Flickr, How to get your videos onto YouTube, demonstrations of P2P software, a tour of MySpace, a tour of Second Life (or let us know if there are other tools or technologies you think people should know about...).

Present a question, problem, or puzzle

Were inviting people from lots of different worlds. We do not expect everybody at Readers Forum to be an expert in some areas. Your expertise is a real resource you may contribute to the Java Jazzup. We want your curiosity to be a resource, too. You can also present a question, problem, or puzzle that revolves around java technologies along with their solution that you think would get really appreciated by the java readers around the globe.

Post resourceful URLs

If you think you know such URL links which can really help the readers to explore their java skills. Even you can post general URLs that you think would be really appreciated by the readers community.

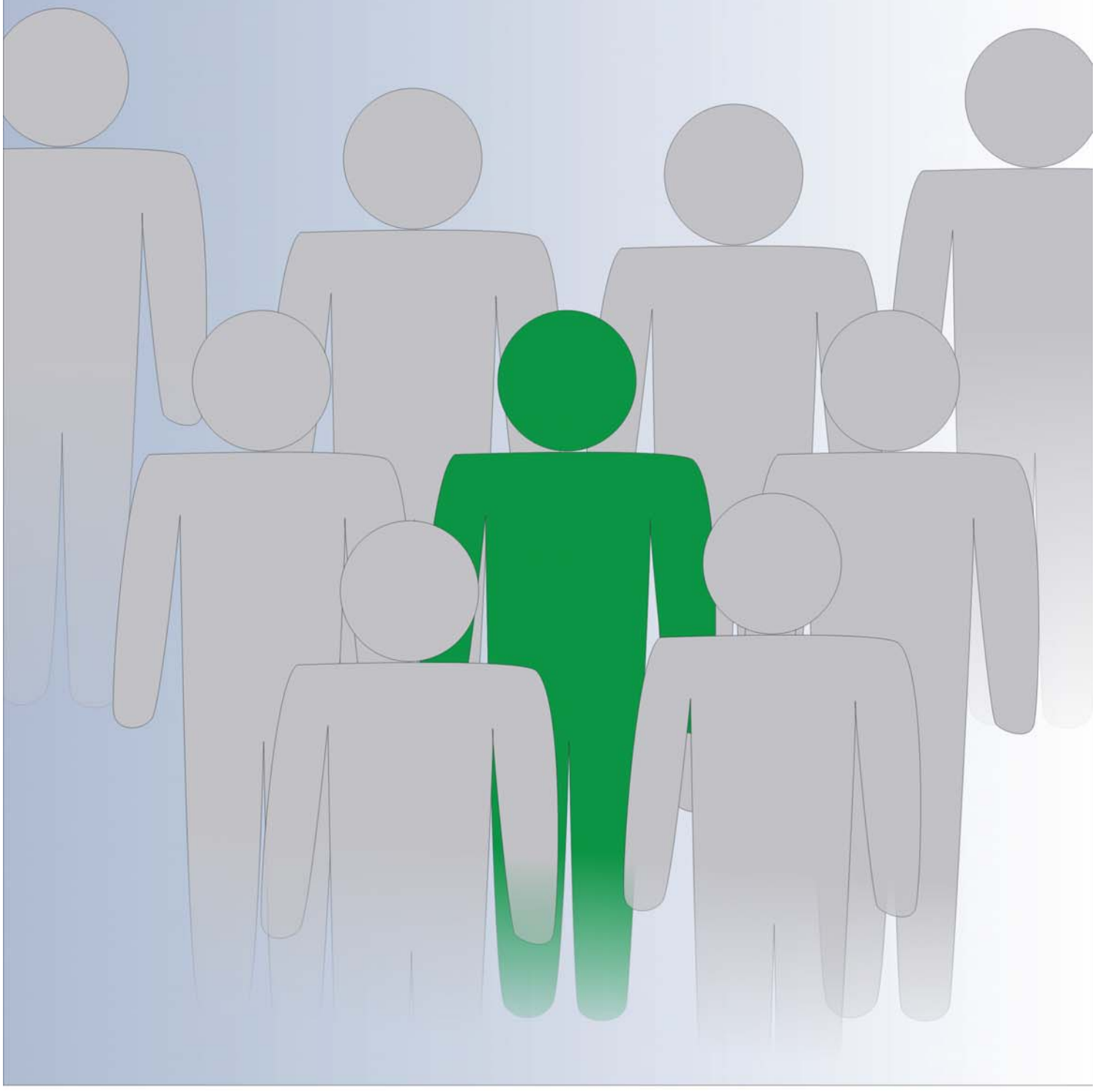
Anything else

If you have another idea for something youd like to do, talk to us. If you want to do something that we havent thought of, have a crazy idea, wed really love to hear about it. Were open to all sorts of suggestions, especially if they promote readers participation.

All Cool Jobs

http://www.allcooljobs.com

BE REGISTERED AND GET A JOB HURRY



NEWSTRACK india

http//www.newstrackindia.com

WHY YOU'LL ALWAYS FIND NEWS IN THE NEWS PAPER

- Society
- World
- Economy
- Sports
- News Week
- Life Style
- Arts Culture
- Entertainment
- Editorial
- Features