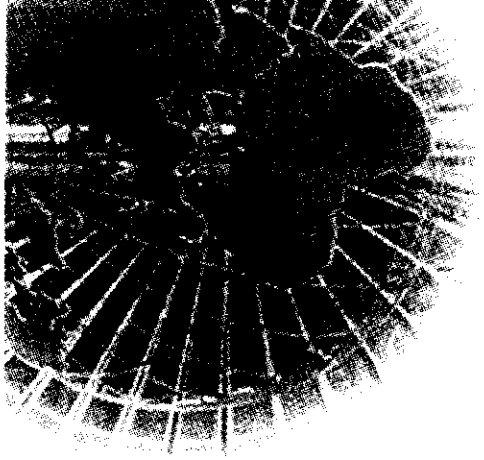


31

Implementing AJAX Frameworks

A framework is considered as a structure according to which software projects are arranged systematically and developed. A framework includes the support programs, code libraries, and a scripting language. In addition to that, it also includes the software that provides help to develop and glue together the different components of your project.

There are a number of AJAX-based frameworks available, but here we are giving information about the important frameworks only. These AJAX Frameworks discussed in this chapter are as follows:

- Jquery Framework
- Prototype Framework
- Script.aculo.us Framework
- DOJO Toolkit Framework
- DWR Framework
- JPSpan Framework
- Rico Framework
- Spry Framework

Let's start discussing each of these frameworks one by one.

Working with Jquery Framework

Jquery was created by John Resig in early 2006. It is a great library for persons who are using JavaScript code and is beneficial for both a beginner and an experienced programmer in JavaScript. Jquery enables developer to code easily. It does not require repeating loops and DOM scripting library calls. Jquery uses least characters to express anything. Thus, the philosophy of Jquery is unique. It is designed to keep the things simple and reusable. You need to download the Jquery Framework from its website `http://www.Jquery.com` and save it on your computer. It is available in both compressed and uncompressed forms. In order to start the working with Jquery, you need a test page in HTML as shown here:

```
<html>
<head>
  <script type="text/javascript" src="/Jquery.js"></script>
  <script type="text/javascript">
  // Javascript code
  </script>
</head>
<body>
  <a href="http://Jquery.com/">Jquery</a>
</body>
</html>
```

The path given in the `src` attribute of `<script>` tag is used only when `Jquery.js` and your HTML file are in the same directory.

NOTE

To get the details of the Jquery Framework, you can visit its website http://www.Jquery.com.

Let's create an AJAX application using the Jquery Framework. The index.html page of Jquery application shows a link that forwards you to the jquey.jsp page. Listing 31.1 shows the code of index.html page (you can find this file in Code\AJAX\Chapter 31\Jquery folder on the CD):

Listing 31.1: The index.html File

```
<html>
<head>
  <link rel="stylesheet" type="text/css" href="style.css">
  <SCRIPT language="JavaScript" src="jquery.js"></SCRIPT>
  <title>Jquery Framework</title>
</head>
```

```
<body>
  <h1> Jquery Framework </h1>

  <a href="jquery.jsp">JQuery</a>
</body>
</html>
```

Figure 31.1 shows the output of index.html page:

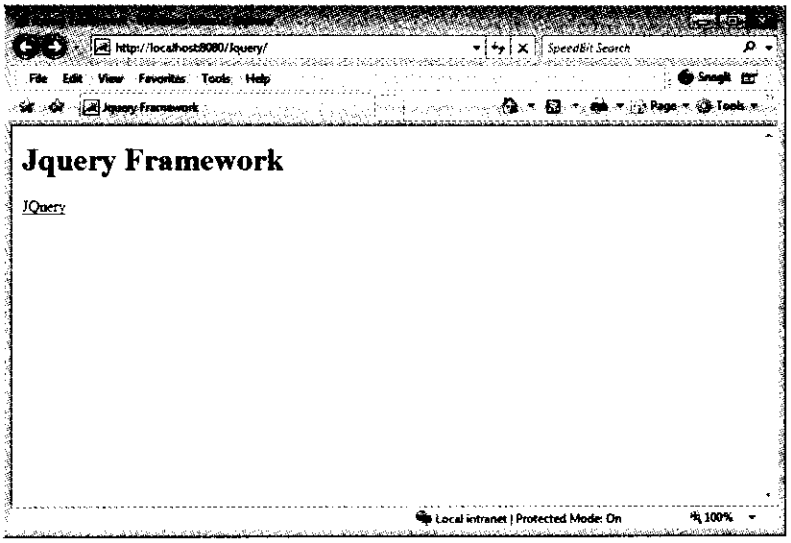


Figure 31.1: Showing Link to Jquery Framework

The application starts with a JSP page, jquery.jsp, which displays when the user clicks on the JQuery hyperlink in Figure 31.1. The code given in Listing 31.2 shows code for the jquery.jsp page (you can find this file in Code\AJAX\Chapter 31\Jquery folder on the CD):

Listing 31.2: The jquery.jsp File

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional
//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>jQuery $.ajax() GET Example</title>
  <script type="text/javascript"src="scripts/jquery.js"></script>
  <script type="text/javascript">
  //<![CDATA[
  function requestCustomerInfo() {
        var sId = $("input#txtCustomerId").val();
        $.ajax({
                type : "GET",
                dataType: "html",
                url : "GetCustomerData.jsp?id=" + sId,
                success : function (oXHR, status) {
                $("div#divCustomerInfo").html(oXHR.responseText);
                alert("Data Retrieved");
                },
                error : function (oXHR, status) {
                $("div#divCustomerInfo").html("An error occurred.");
                }
        });
```

```
    }
    //]]>
</script>
</head>
<body>
    <H2>Enter customer ID</H2>
    <p>Customer ID: <input type="text" id="txtCustomerId" value="" /></p>
<p><input type="button" value="Get Customer Info" onclick="requestCustomerInfo()" /></p>
    <div id="divCustomerInfo"></div>
</body>
</html>
```

The jquery.jsp page simply prompts the user to enter the ID of the customer to retrieve his/her information. The Jquery Framework accesses the input field and evaluates its value by using the following syntax:

```
var sId = $("input#txtCustomerId").val();
```

Next, the framework sends a GET request to GetCustomerData.jsp page to retrieve information from the database. The request is sent by the $.ajax method of Jquery as shown here:

```
$.ajax({
    type : "GET",
    dataType: "html",
    url : "GetCustomerData.jsp?id=" + sId,
    success : function (oXHR, status) {
$("div#divCustomerInfo").html(oXHR.responseText);alert("Data Retrieved");
    },
    error : function (oXHR, status) {
            $("div#divCustomerInfo").html("An error occurred.");
    }
});
```

The url parameter tells which resource is requested from the server and the type parameter tells the type of the request (GET, POST). The success function executes when the status = 200 and it displays the response, otherwise the error function executes.

The GetCustomerData.jsp page accesses the record of customer from the database. The code given in Listing 31.3 shows code for the GetCustomerData.jsp page (you can find this file in Code\AJAX\Chapter 31\Jquery folder on the CD):

Listing 31.3: The GetCustomerData.jsp File

```
<%@page contentType="text/plain"%>
<%@page pageEncoding="UTF-8"%>
<%@ page import="java.sql.*" %>
<%!protected String getCustomerData(int id){
  try {
        Connection con = com.kogent.ajax.DatabaseConnector.getConnection();
        System.out.println("Accessing all Customers with ID = : " +id);
        String sql1 = "Select * from Customer where CUSTOMERID="+id;
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(sql1);
        boolean found = rs.next();
        StringBuffer buffer = new StringBuffer();
        System.out.println("Query Executed : " +sql1);
        if (found) {
              buffer.append(rs.getString("Name"));
              buffer.append("<br />");
              buffer.append(rs.getString("Address"));
              buffer.append("<br />");
              buffer.append(rs.getString("City"));
              buffer.append("<br />");
              buffer.append(rs.getString("State"));
              buffer.append("<br />");
```

```
                buffer.append(rs.getString("pin"));
                buffer.append("<br /><br />");
                buffer.append("Phone: " + rs.getString("Phone"));
                buffer.append("<br /><a href=\"mailto:");
                buffer.append(rs.getString("Email"));
                buffer.append("\">");
            } else {
                buffer.append("Customer with ID ");
                buffer.append(id);
                buffer.append(" could not be found.");
            }
            rs.close();
            con.close();
            System.out.println("Resulted String : " +buffer.toString());
            return buffer.toString();
        } catch (Exception ex){
            System.out.println("Error Comes Terminated : " +ex);
            return "Error in Getting Customers Info";
        }
    }%>
    <%
    String id = request.getParameter("id");
    String callback = request.getParameter("callback");
    String message = "";
    int customerId = -1;
    try {
      customerId = Integer.parseInt(id);
      System.out.println("You Entered: " +customerId);
      message = getCustomerData(customerId);
    } catch (Exception ex) {
      message = "Invalid customer ID.";
    }
    %>
    <%=message%>
```

The GetCustomerData.jsp page uses JDBC API for accessing the database.

In Listing 31.3, the connection is created and the data from the customers table is retrieved based on the customer id entered by the user. Here's the code snippet that provides the DDL queries for creating the customer table:

```
create table customer(
customerid int primary key,
name varchar(30),
address varchar(30),
city varchar(25),
state varchar(25),
pin varchar(6),
phone varchar(15),
email varchar(20));
```

Now insert a row in the table customer using following SQL command:

```
insert into customer values (3,'Vikash KumarSuman',' H.No 620 D/A-15','South Ext',' New
  Delhi','110030','9311679802','vikash.tc@gmail.com');
```

The value entered by the user from the Jquery.jsp page is first converted into int and then sent to the `getCustomerData()` function.

The output of Jquery.jsp page is shown in Figure 31.2:

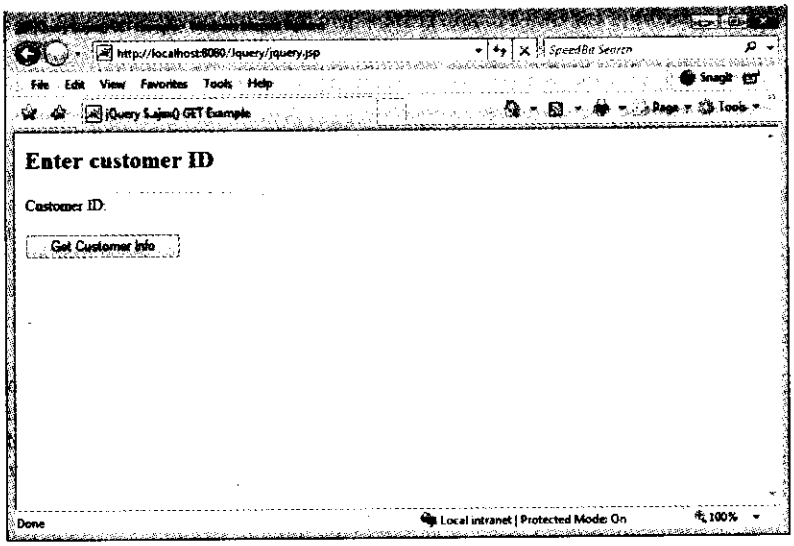


Figure 31.2: Showing Jquery.jsp Page

When the user enters the customer ID and clicks the Get Customer Info button, it displays the Data Retrieved message, as shown in Figure 31.3:

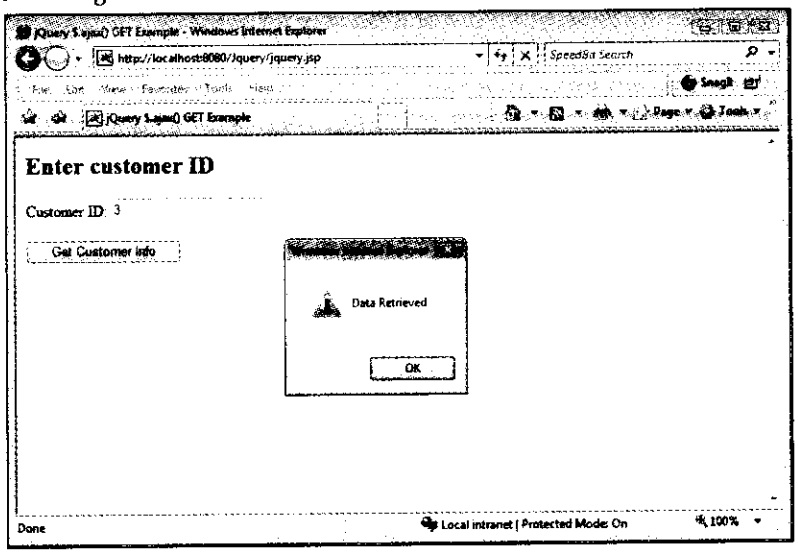


Figure 31.3: Showing Data Retrieved Message

The alert message shows that the record is retrieved from the database. This alert message is defined in the success function of the $ajax () function. Next, we are discussing the Prototype Framework.

Working with Prototype Framework

The Prototype Framework is a JavaScript Framework, which provides an AJAX Framework and other utilities. Ruby on Rails integrates the Prototype Framework. Prototype provides various functions for developing JavaScript applications. The features range from programming shortcuts to major functions for dealing with `XMLHttpRequest`. You can download this framework from the website `http://www.prototypejs.org`. If

you want to refer to an element of the DOM of an HTML page, the usual function identifying an element will be as follows:

```
document.getElementById("id_of_element")
```

By using the Prototype Framework, we can reduce the code as shown in following code snippet:

```
$("id_of_element")
```

Therefore, you can see that the code becomes much simpler by using the Prototype Framework. Similar to the `$()` function, we have the `$F()` function. The `$F()` function returns the value of the requested form element. The function will return the data contained in the element for "text" input, while for a select input element, the function will return the currently selected value:

```
$F("id_of_input_element")
```

This framework allows you to handle AJAX calls in an easy manner and the entire AJAX functionality is encapsulated in `Ajax` object.

In order to reduce the amount of code needed to run a cross-browser XMLHttpRequest function, the Prototype's AJAX object provides an easy way of invoking the function without depending on the specific Web browsers. There are two forms of AJAX object. The first is `Ajax. Request`, which returns the raw XML output from an AJAX call. The other AJAX object is `Ajax. Updater` which updates contents of DOM element by the response returned by AJAX request.

The following `Ajax.Request` finds the values of two HTML value inputs, requests a page from the server by using the values as `POST` values, and then runs a custom function called `showResponse()` when complete. The following code snippet illustrates the use of `Ajax. Request` method:

```
var value1 = escape($F("name_of_id_1"));
var value2 = escape($F("name_of_id_2"));
var url = "http://yourserver/path/server_script";
var pars = "value1=" + value1 + "&value2=" + value2;
var myAjax = new Ajax.Request(
  url, {
        method: "post",
        parameters: pars,
        onComplete: showResponse
  });
```

NOTE

You can get further details of the Prototype Framework from the website http://www.prototypejs.org.

Let's create a web application using the Prototype Framework. The index.html page of Prototype application shows a link that forwards you to the Prototype.jsp page. Listing 31.4 shows the code of index.html page (you can find this file in Code\AJAX\Chapter 31\Prototype folder on the CD):

Listing 31.4: The index.html File

```
<html>
<head>
  <link rel="stylesheet" type="text/css" href="style.css">
  <SCRIPT language="JavaScript" src="prototype.js"></SCRIPT>
  <title>Jquery Framework</title>
</head>
<body>
  <h1> Prototype Framework </h1>

  <a href="Prototype.jsp">Prototype</a>
</body>
</html>
```

Figure 31.4 shows the output of index.html page:

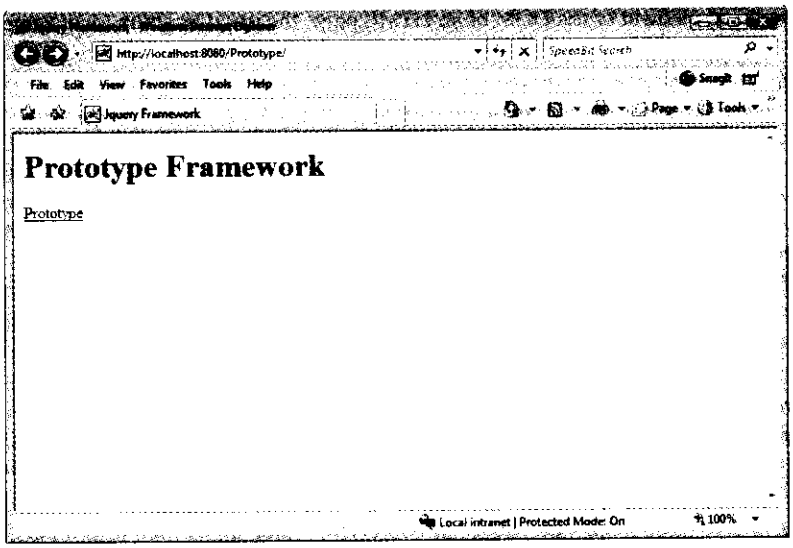


Figure 31.4: Showing Link to Prototype Framework

The application starts with a JSP page, Prototype.jsp, which displays when the user clicks on the "Prototype" hyperlink in Figure 31.4. The page uses the prototype.js file, which you can find in the scripts folder of the Prototype application. The code given in Listing 31.5 shows code for Prototype.jsp page (you can find this file in Code\AJAX\Chapter 31\Prototype folder on the CD):

Listing 31.5: The Prototype.jsp File

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Implementing Prototype</title>
  <script type="text/javascript"src="scripts/prototype.js"></script>
  <script type="text/javascript">
  //<![CDATA[
  function requestCustomerInfo() {
        var sId = document.getElementById("txtCustomerId").value;
        var oOptions = {
              method: "get",
              parameters: "id=" + sId,
              onSuccess: function (oXHR, oJson) {
                    displayCustomerInfo(oXHR.responseText);
              },
              onFailure: function (oXHR, oJson) {
              displayCustomerInfo("An error occurred: " + oXHR.statusText);
              }
        };
        var oRequest = new Ajax.Request("GetCustomerData.jsp", oOptions);
  }
  function displayCustomerInfo(sText) {
        var divCustomerInfo = document.getElementById("divCustomerInfo");
        divCustomerInfo.innerHTML = sText;
  }
  //]]>
  </script>
```

```
</head>
<body>
  <h2>Enter customer ID</h2>
  <p>Customer ID: <input type="text" id="txtCustomerId" value="" /></p>
<p><input type="button" value="Get Customer Info" onclick="requestCustomerInfo()" /></p>
  <div id="divCustomerInfo"></div>
</body>
</html>
```

The Prototype.jsp page simply prompts the user to enter the ID of the customer to retrieve his/her information. The Prototype Framework accesses the input field and evaluates its value by using the following syntax:

```
var sId = document.getElementById("txtCustomerId").value
```

Next, the framework sends a GET request to the GetCustomerData.jsp page to retrieve information from the database. The request is sent by the Ajax.Request() method of Prototype as shown here:

```
var oRequest = new Ajax.Request("GetCustomerData.jsp", oOptions);
```

In the preceding code snippet, the `oOptions` parameter defines the type of request and the `onSuccess` and `onFailure` method. The requested resource is GetCustomerData.jsp, which is shown in Listing 31.3. The view created by the Prototype.jsp page is same as the one shown in Figure 31.2. When the user enters customer ID 3, it displays the record of customer as shown in Figure 31.5:

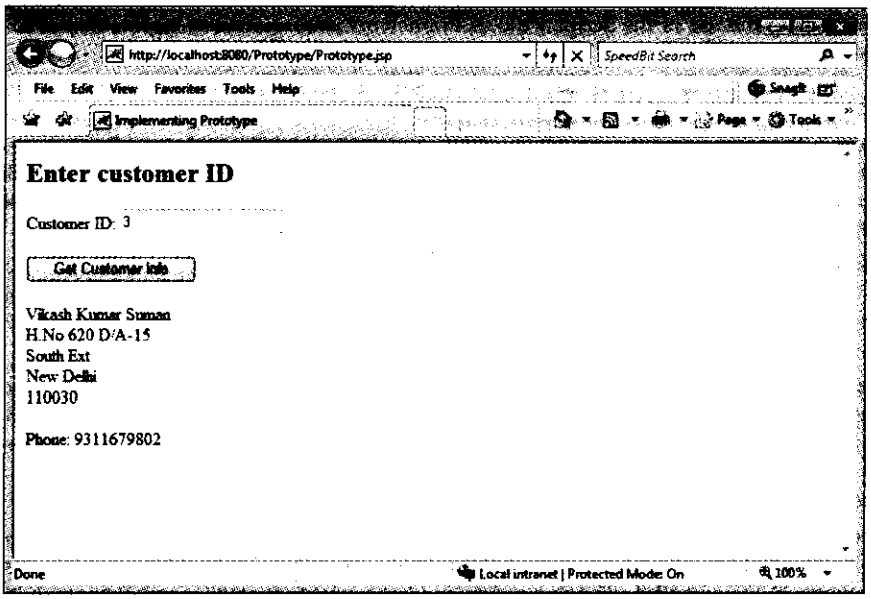


Figure 31.5: Displaying Information related with Customer ID 3

The record is fetched from the database when the user enters the customer ID and clicks the "Get Customer Info" button, as shown in Figure 31.5.

In this section you have learned using the Prototype Framework to create an AJAX application. Let's discuss using the Script.aculo.us Framework, in the following section.

Working with Script.aculo.us Framework

Script.aculo.us is a JavaScript library that makes use of Prototype library to develop rich user interfaces. Prototype smoothes out many of the wrinkles of Ajax development. But when the Prototype is used with Script.aculo.us, it transforms the way we approach the web user interface by making features, such as animation and dragging and dropping as simple as a few lines of code. Like Prototype, Script.aculo.us covers several distinct areas.

Script.aculo.us makes it easy to provide visual effects in your page through its Effects library. This library is remarkable because it enables a wide range and quality of effects within your web page. This good design is made possible by the language features provided by Prototype. Dragging and dropping is a common user interface operation in desktop applications, and in many cases it provides the most convenient and intuitive way of interacting with a computer. The HTML DOM has no direct support for drag-and-drop events, and implementing drag and drop in JavaScript means relying on nothing more than mouse click and movement events. Script.aculo.us implements the drag-and-drop feature that can be applied to most types of DOM elements with relatively little code. The look and feel of the user's interaction can be customized using the Effects library, and custom event handlers provide callbacks for all stages of the drag-and drop event.

Script.aculo.us also provides a number of complete controls in the Components library. The Components library contains controls, such as Autocompleter control, in-place editor control, and slider control. The Autocompleter can attach a drop-down element to any text field, which can provide an ordinary HTML Form element with features similar to Google Suggest; and the in-place editor allows any DOM element to transform itself into a text input field and back again.

NOTE

To get the details of the Script.aculo.us Framework, you can visit http://script.aculo.us/

After understanding the use of the Script.aculo.us Framework, let's discuss using the Dojo Toolkit Framework, in the following section.

Working with Dojo Toolkit Framework

Dojo provides a lot of power and attempts in order to make it digestible in layers. For server-side developers there is "widgets without coding", for HTML and CSS developers Dojo provides wonderful facilities for quickly building template driven widgets, and for serious JavaScript and DHTML hackers Dojo is the standard library you will wish JavaScript always had.

Dojo is an Open Source DHTML toolkit written in JavaScript. It builds on several contributed code bases (nWidgets, f(m) and Burstlib), which is why we refer to it sometimes as a "unified" toolkit. Dojo aims to solve some long-standing historical problems with DHTML which prevented mass adoption of dynamic Web application development.

Dojo is a set of layered libraries. The bottom most layer is the **packaging system** that enables you to customize the distribution of Dojo for your application. On top of Dojo's package, the system resides in language libraries, such as the Dojo **event system** and the **language utilities** that greatly improve and simplify the lives of JavaScript developers. Environment-specific libraries are provided in some cases, but in most cases, you'll only need to know that all of Dojo works without incident across every major browser.

The bulk of the Dojo code lives in the application support libraries, which are too numerous to display completely in the diagram. The `dojo.gfx` provides native vector graphics support across the major browser implementations of SVG and VML. The `dojo.lfx` is a lightweight effects library, while `dojo.io` is "where the AJAX lives."

Most of the "action" in Dojo is in the **widget** toolkit that contains a template-driven system for constructing widgets, a declarative parser for putting widgets in pages without writing JavaScript, and a set of base components that implement common-case solutions to web interaction problems that HTML+CSS alone cannot solve.

Dojo allows you to easily build dynamic capabilities into web pages and any other environment that supports JavaScript sanely. You can use the components that Dojo provides to make your web sites more useable, responsive, and functional. With Dojo, you can build degradable user interfaces more easily, prototype interactive widgets quickly, and animate transitions. You can use the lower-level APIs and compatibility layers from Dojo to write portable JavaScript and simplify complex scripts. Dojo's event system, I/O APIs, and generic language enhancement form the basis of a powerful programming environment. You can use the Dojo build

tools to write command-line unit-tests for your JavaScript code. The Dojo build process helps you to optimize your JavaScript for deployment by grouping sets of files together and reusing those groups through "profiles."

Dojo does all of these things by layering capabilities onto a very small core that provides the package system and little else. When you write scripts with Dojo, you can include as little or as much of the available APIs as you need to suit your needs. Dojo provides the following:

- **Multiple Points of Entry**—You can start using Dojo at the level you are most comfortable with. For example, expert JavaScript programmers can use the foundation capabilities to be more productive quickly, while Web designers and developers can use a set of easy to use, modify, and extend components that make their applications more responsive without requiring them to learn a large JavaScript API. This fundamental design decision drives the layered implementation of most of the major capabilities of Dojo.
- **Interpreter Independence**—Dojo is squarely a JavaScript toolkit but, within the realm of JavaScript interpreters and environments, not everything was created equally. Dojo supports, at least, the very core of the system on as many JavaScript enabled platforms as possible. This allows Dojo to serve as a "standard library" for JavaScript programmers as they move between client-side, server-side, and desktop programming environments.
- **Forward Looking APIs**—No one has a crystal ball when it comes to what technologies will be broadly available or used in five years, but Dojo attempts to provide APIs that are generic enough to be (directly) useful with today's capabilities while still building in room for future improvement. The `dojo.io.bind()` interface is a great example of this principle. When first written it wrapped only a single Transport class, but now it provides a normalized interface to many ways of receiving and sending data from JavaScript enabled environments.
- **ReducingBarriersToAdoption**—This core philosophy behind Dojo's design acknowledges the fact that there is no need to work with tools that are difficult to use, no matter how good they are. According to this philosophy, Dojo should be built in every way (licensing to deployment) which do not give users any reason for not trusting or use Dojo for the tasks it's good at. Many of the project's overall decisions get made on the basis of this principle.

Dojo is being built around a single markup language that provides application authors a (more) simple way of declaring and using responsive DHTML interface components. Renderings can be made available in several rendering contexts (such as SVG, or perhaps even the desktop or Flash), but the markup language (DojoML) and scripting language (JavaScript) will not change. Better yet, the DojoML parser accepts extended HTML and SVG as valid input, and can be used to easily create Degradable Responsive Applications.

NOTE

You can get more information about Dojo from http://dojotoolkit.org.

Let's create a web application using the Dojo Framework. The index.html page of Dojo application shows a link that forwards you to the Dojo.html page. Listing 31.6 shows the code of index.html page (you can find this file in Code\AJAX\Chapter 31\Dojo folder on the CD):

Listing 31.6: The index.html File

```
<html>

<head>
  <link rel="stylesheet" type="text/css" href="style.css">
  <SCRIPT language="JavaScript" src="dojo.js"></SCRIPT>
  <title>Dojo Framework</title>
</head>

<body>
  <h1> Dojo Framework </h1>

  <a href="Dojo.html">Dojo</a>
</body>
```

```
</html>
```

Figure 31.6 shows the output of index.html page:

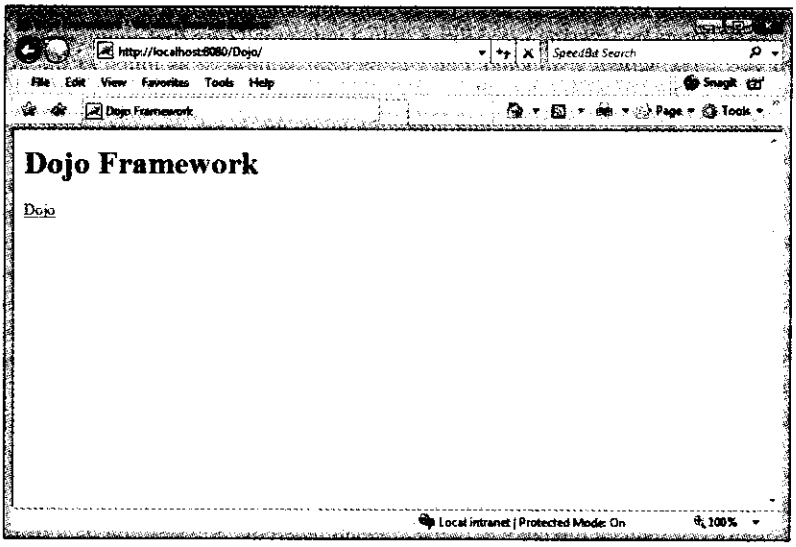


Figure 31.6: Showing Link to Dojo Framework

The application starts with a HTML page, dojo.html, which displays when the user clicks on the Dojo hyperlink in Figure 31.6. The page uses the dojo.js file, which you can find in the scripts folder of the application. The code given in listing 31.7 shows code for the Dojo.html page (you can find this file in Code\AJAX\Chapter 31\Dojo folder on the CD):

Listing 31.7: The Dojo.html File

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <meta name="generator" content="HTML Tidy, see www.w3.org">
  <title>Implementing Dojo</title>
  <script language="JavaScript" src="scripts/dojo.js"></script>
  <script language="JavaScript" src="scripts/dojo_p.js"></script>
  <SCRIPT language="JavaScript" type="text/javascript">
  function retrieveUserPassword() {
  id=document.getElementById("uid");
        dojo.io.bind({
                url: "/Dojo/userid?id=" + escape(id.value),
                load: function(type, data, evt){
                        displayUserAndPassword(data); },
                error: function(type, data, evt){
                        reportError(type,data,evt); },
                mimetype: "text/plain"
        });
  }
  function displayUserAndPassword(data){
        var jsonData = data;
        var myJSONObject = eval('(' + jsonData + ')');
        if(myJSONObject.userpass.username== null)
        {
        alert('no entry in database for id: '+myJSONObject.userpass.id);
        var user = document.getElementById('user').value="";
        var user = document.getElementById('password').value="";
```

```
        }
        else
        {
                var user = document.getElementById

                ('user').value=myJSONObject.userpass.username;
                var user = document.getElementById

                ('password').value=myJSONObject.userpass.password;
        }
  }
  function reportError(type,data,evt) {
        alert('error retrieving a Username and Password for that Id.');
  }
  </script>
</head>
<body>
<h3>Enter User Id and it will retrieve Username and Password for the User</h3>
<table align="left" class="borderless">
<tr>
  <td colspan="3">
  <table align="left" class="borderless">
  <tr>
        <td class="headtext">
        </td>
  </tr>
  </table>
  </td>
</tr>
<tr>
  <td>User ID:</td>
  <td align="left">
  <input type="text" id="uid" name="uid" onblur="retrieveUserPassword()" >
  </td>
</tr>
<tr>
  <td>Username:</td>
  <td align="left"><input type="text" id="user" name="user"></td>
</tr>
<tr>
  <td>Password:</td>
  <td align="left"><input type="text" id="password" name="password"></td>
</tr>
</table>
</body>
</html>
```

The page prompts the user to enter the User Id and after that it retrieves the username and password for the entered User Id. The page then sends a request to the UserPasswordServlet Servlet, which retrieves the username and password for the entered User Id from the database. The code given in Listing 31.8 shows code for the UserPasswordServlet.java file (you can find this file in Code\AJAX\Chapter 31\Dojo\src folder on the CD):

Listing 31.8: The UserPasswordServlet.java File

```
package com.kogent.ajax.servlet;

import java.io.IOException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
```

```
import java.sql.Statement;
import java.util.HashMap;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.kogent.ajax.DatabaseConnector;
import com.kogent.ajax.JSONUtil;

public class UserPasswordServlet extends HttpServlet
{

  private static final long serialVersionUID = 1L;

  public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
  {
        String responseString = null;

       String id = req.getParameter("id");
       if (id != null)
       {
           HashMap userpass = getUserPassword(id);
           responseString = JSONUtil.buildJSON(userpass, "userpass");
       }
       if (responseString != null)
       {
           res.setContentType("text/xml");
           res.setHeader("Cache-Control", "no-cache");
           res.getWriter().write(responseString);
       } else
       {
           // If key comes back as a null, return a question mark.
           res.setContentType("text/xml");
           res.setHeader("Cache-Control", "no-cache");
           res.getWriter().write("?");
       }
  }

   private HashMap getUserPassword(String userId)
   {
       Connection con = DatabaseConnector.getConnection();
       HashMap userPasswordMap = new HashMap();
       userPasswordMap.put("id", userId);
       String queryString = "";
       try
       {
           queryString = "SELECT username, password FROM userpassword where id="+userId;
           Statement select = con.createStatement();
           ResultSet result = select
                   .executeQuery(queryString);

           while (result.next())
           {
               String username;
               String password;

               username = result.getString("username");
               if (result.wasNull())
```

```
                {
                    username = "";
                }
                userPasswordMap.put("username", username);
                password = result.getString("password");
                if (result.wasNull())
                {
                    password = "";
                }
                userPasswordMap.put("password", password);
            }
        } catch (Exception e)
        {
            System.out.println("exception caught getting username/password:"+queryString+"
         "+e.getMessage());
        } finally
        {
            if (con != null)
            {
                try
                {
                    con.close();
                } catch (SQLException e) {
                }
            }
        }
        return userPasswordMap;
    }
}
```

The "UserPasswordServlet" connects with the database using the DatabaseConnector.java file. The DatabaseConnector.java file contains JDBC statements for connecting with the database. The source code for this file is given in listing 31.9 (you can find the file DatabaseConnector.java in Code\AJAX\Chapter 31\Dojo\src folder on the CD):

Listing 31.9: The DatabaseConnector.java File

```
package com.kogent.ajax;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class DatabaseConnector
{
  public static Connection getConnection()
  {
        Connection con = null;
        String driver = ("com.mysql.jdbc.Driver");
        try
        {
            Class.forName(driver).newInstance();
        } catch (Exception e) {
            System.out.println("Failed to load MySQL driver.");
            return null;
        }
        try
        {
            con = DriverManager
            .getConnection("jdbc:mysql://localhost:3306/emp","root","password");
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
            return con;
        }
    }
```

The Servlet, "UserPasswordServlet" interacts with a table named userpassword. The userpassword table can be created by using following SQL command:

```
create table userpassword (id varchar(5),username varchar(25),password varchar(25));
```

Now we insert two rows in this table using following SQL commands:

```
insert into userpassword values('1','vikash','vikash');
insert into userpassword values('2','Suchita','suchita');
```

The UserPasswordServlet Servlet retrieves the Username and Password for the entered user Id from the database table named userpassword and then builds JSON data using the JSONUtil.java helper class, which converts the HashMap data into JSON data. The code given in Listing 31.10 shows code for the JSONUtil.java file (you can find this file in Code\AJAX\Chapter 31\Dojo\src folder on the CD):

Listing 31.10: The JSONUtil.java File

```
package com.kogent.ajax;

import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

public class JSONUtil
{

    public static String buildJSONArray(List list, String title)
    {
        StringBuffer returnJSON = new StringBuffer("\r\n{\"" + title + "\": [");
        String key = "username";
        String value = "";
        Iterator it = list.iterator();

        while (it.hasNext())
        {
            value = (String) it.next();
            returnJSON.append("\r\n{\"" + key + "\": \"" + value + "\"},");
        }
        // remove the last comma
        int lastCharIndex = returnJSON.length();
        returnJSON.deleteCharAt(lastCharIndex - 1);
        returnJSON.append("\r\n]}");
        return returnJSON.toString();
    }
    public static String buildJSON(HashMap map, String title)
    {
        StringBuffer returnJSON = new StringBuffer("\r\n{\"" + title + "\":[");
        String key = "";
        String value = "";

        Iterator it = map.entrySet().iterator();

        while (it.hasNext())
        {
            Map.Entry e = (Map.Entry) it.next();
            value = (String) e.getValue();
            key = (String) e.getKey();
            returnJSON.append("\r\n\"" + key + "\": \"" + value + "\",");
```

```
        }
        int lastCharIndex = returnJSON.length();
        returnJSON.deleteCharAt(lastCharIndex - 1);
        returnJSON.append("\r\n}}");
        return returnJSON.toString();
    }
}
```

The view, created by the Dojo.html page is shown in Figure 31.7. When you enter the User Id 1, it displays the username and password of the user from the database, as shown in Figure 31.7:

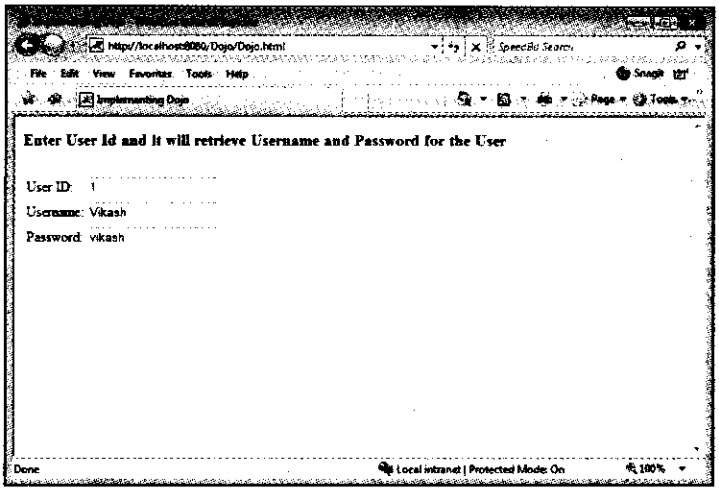


Figure 31.7: Retrieving Username and Password using Dojo

In this section, you have learned implementing the Dojo Framework to create AJAX application. Let's discuss another framework: the DWR Framework in detail, in the following section.

Working with DWR Framework

DWR is an excellent way to harness the power of Java for use in Web applications. Joe Walker, the primary developer of DWR, constantly updates the framework with new functionality designed to make Web applications more responsive. Since DWR works with literally hundreds of available Java packages, chances are that there's something already available that can perform the operations necessary for your application. There is also the advantage that there is no shortage of information, tutorials, and experts willing to share their knowledge about the Java platform.

Direct Web Remoting (DWR) is an AJAX Framework for Java/JSP available for download at `http://getahead.ltd.uk/dwr`. DWR works similar to JPSpan. In that it uses Java's version of reflection to examine Java bean classes and then create JavaScript wrappers to call the various methods. Also, like JPSpan, DWR includes its own JavaScript library for cross-browser Ajax communication, freeing the developer from worrying about browser incompatibilities. DWR assumes the use of Apache Tomcat (`http://tomcat.apache.org`).

NOTE

DWR expects that the classes you use will be Java beans, meaning that they can be created without passing any information to the constructor. This is important because the server-side objects don't persist from request to request and need to be created from scratch each time.

Setting up DWR in your Web application is very simple. The first step is to place the `dwr.jar` file into the `WEB-INF/lib` directory. The next step is to edit the `web.xml` file contained in `WEB-INF`. There are two sections that need to be added.

The first section describes the DWR invoker Servlet:

```
<servlet>
<servlet-name>dwr-invoker</servlet-name>
<display-name>DWR Servlet</display-name>
<servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
<init-param>
<param-name>debug</param-name>
<param-value>true</param-value>
</init-param>
</servlet>
```

This code needs to go with the other `<servlet/>` tags inside of `web.xml`. The second section that needs to be added is as follows:

```
<servlet-mapping>
<servlet-name>dwr-invoker</servlet-name>
<url-pattern>/dwr/*</url-pattern>
</servlet-mapping>
```

This information provides the URL that can be used to call the DWR invoker and must be located alongside any other `<servlet-mapping/>` tags in the file.

The final step is to create a file called `dwr.xml` in the `WEB-INF` directory. This file specifies the Java classes that should be wrapped by DWR for use on the client. A sample of dwr.xml file is shown here:

```
<!DOCTYPE dwr PUBLIC
"-//GetAhead Limited//DTD Direct Web Remoting 1.0//EN"
"http://www.getahead.ltd.uk/dwr/dwr10.dtd">
<dwr>
<allow>
<create creator="new" javascript="JDate">
<param name="class" value="java.util.Date"/>
</create>
<create creator="new" javascript="Demo">
<param name="class" value="your.java.Bean"/>
</create>
</allow>
</dwr>
```

This is the example file suggested on DWR's web site. The root element, `<dwr/>`, contains an `<allow/>` element. Each of the `<create/>` elements contained within specify objects that are allowed to be created by JavaScript. In each `<create/>` element, the `javascript` attribute specifies the name that must be used in JavaScript to invoke the Java bean. Since there is already a Date object in JavaScript, this example specifies the constructor name JDate for using the Java Date object. The `<param/>` element specifies the complete class name for the bean to use. You can add any number of `<create/>` elements to allow JavaScript wrappers for your custom beans. After making these changes, you can go to the following URL to test the installation:

```
http://hostname/webappName/dwr/
```

After discussing the DWR Framework, let's discuss the JPSpan Framework, which is similar to the DWR Framework, in the following section.

Working with JPSpan Framework

JPSpan is an AJAX Framework built with the intention of integrating client-side JavaScript with server-side code written in PHP. PHP has been around since the mid-nineties and has grown from simple beginnings to a full-fledged object-oriented language that can run on both Windows and UNIX/Linux platforms. The main advantage of using PHP over other platforms, such as Java or .NET is that it is smaller, much simpler to install, and more lightweight, needing only a fraction of the memory of the Java runtime or the .NET CLR.

JPSpan works by analyzing a PHP class by using reflection. It then emits a number of JavaScript methods that accepts the same arguments as the methods of the class on the server. When one of these methods is called, it uses a cross-browser library to instantiate the appropriate `XmlHttpRequest` for the platform it is running on,

and posts the data to the server. The response is then read and passed to a function on the client for further use within the page.

NOTE

Reflection in this case means examining a class to find out its properties and methods. It is a commonly available technique in object-oriented languages and is often used as a way of writing generic code that can work with a large number of classes.

The first thing you need is a Web server that supports PHP, version 5. This can be the ubiquitous Apache Web server for UNIX/Linux machines or IIS for Windows (although others also support PHP, these are the most common). When installing with IIS, there is a choice of running the PHP processor as an out of process server by using CGI or in process by using ISAPI. The first method is recommended for reliability and the second for performance.

NOTE

You can find the PHP installation, along with a list of compatible Web servers, at http://uk.php.net. There is a comprehensive set of instructions with a list of Frequently Asked Questions detailing how to resolve some common difficulties and problems.

The actual JPSpan installation can be found at `http://sourceforge.net/projects/jpspan`. Aside from the documentation and some examples, it consists of a number of PHP pages alongside several JavaScript files. The entry point, `JPSpan.php`, sits in a folder named JPSpan, which is best placed directly under the root of your Web server (for IIS this is typically `C:\InetPub\wwwroot` and for Apache it is `/usr/local/apache/htdocs`). There is a subfolder, also called JPSpan that contains additional PHP and JavaScript files.

There are two stages involved in developing the server-side page, i.e. implementing the business classes that will do the actual processing and hooking them into the framework so that they become accessible from the client.

The first stage is to create a standard server-side skeleton page that can be used as a starting point for most JPSpan projects. This consists of basic plumbing needed to inform JPSpan of the business class as well as other options, such as what to do in the event of an error. The second stage is the coding of the business class or classes to be used, which requires the knowledge of PHP and the syntax used to declare classes. The syntax is very similar to C++ and Java, so any knowledge of those two languages will help.

After the business class has been tested outside of the JPSpan Framework, it can be incorporated into the skeleton page ready for use by the client.

The basic structure of a server-side processing page looks something like the one shown here:

```
<?php
//Business class to be included here
class Customer
{
  //Customer class implementation to be included here
}
// Including this sets up the JPSPAN constant
require_once '../JPSpan/JPSpan.php';
// Load the PostOffice server
require_once JPSPAN . 'Server/PostOffice.php';
// Create the PostOffice server
$PostOffice = & new JPSpan_Server_PostOffice();

// Register the Customer class with it…
$PostOffice->addHandler(new Customer());
// This allows the JavaScript to be seen by
// just adding ?client to the end of the
```

```
// server's URL
if (isset($_SERVER['QUERY_STRING']) && strcasecmp
($_SERVER['QUERY_STRING'],
'client') == 0)
{
  // Compress the output Javascript feature
  // turn this off it has performance problems
  define('JPSPAN_INCLUDE_COMPRESS', false);
  // Display the Javascript client
  $PostOffice->displayClient();
}
else
{
  // This is where the real serving happens..
  // Include error handler
  // PHP errors, warnings and notices serialized to JS
  require_once JPSPAN. 'ErrorHandler.php';
  // Start serving requests..
  $PostOffice->serve();
}
?>
```

Following the opening PHP tag is the place where the code for the actual class will be included. This can be actually in the page or can be held in a separate file and accessed through one of the PHP include mechanisms:

```
<?php
//Business class to be included here
class Customer
{
  //Customer class implementation to be included here
}?>
```

After discussing the JPSpan Framework, let's discuss the Rico Framework in detail, in the following section.

Working with Rico Framework

Rico is another AJAX Framework, which is based on JavaScript and provides the functionality of drag-and-drop. It is based on Prototype Framework, which is already discussed in this chapter. Rico is provided open source for either personal or commercial use. You can use Rico for building Accordion component, which are found in Macromedia Flex and Laszlo. You can design a very rich interface by using Rico's style. These styles provide many Cinematic effects as well as some simple visual style effects. We know that desktop applications provide drag-and-drop functionality so that the user simply interacts with the application. Rico provides simple interfaces for enabling your Web application to support drag-and-drop features. This can be done by registering any HTML element or JavaScript object as draggable and any other HTML element or JavaScript object as a drop zone and the rest of the work is done by Rico. Rico provides a simple interface for registering Ajax request handlers, HTML elements, and JavaScript objects as Ajax response objects. In Rico, many elements and JavaScript objects may be updated as a result of one AJAX request.

To start development with Rico, download the latest version of Rico from http://openrico.org/downloads. As Rico Framework depends on the Prototype library, you start your Rico application by importing two JavaScript libraries—prototype.js and rico.js.

You can find these libraries in the downloaded binary of Rico. This can be done as follows:

```
<script language="JavaScript" src="scripts/prototype.js"></script>
<script language="JavaScript" src="scripts/rico.js"></script>
```

Next you need to register a request handler with the Rico's ajaxEngine. The ajaxEngine sends the AJAX request to the server. This can be done as follows:

```
ajaxEngine.registerRequest ('myRequestHandle', 'rico');
```

The first parameter of `registerRequest ()` function defines the name of the request handle, which we are using to make the Ajax request. The second parameter is the string 'rico', which is registered with the ajaxEngine.

Next you need to register your AJAX elements by using the registerAjaxElement () function of ajaxEngine. Here's an example in which we register three Ajax elements:

```
ajaxEngine.registerAjaxElement('firstDiv');
ajaxEngine.registerAjaxElement('secondDiv');
ajaxEngine.registerAjaxElement('third');
```

While you are performing drag-and-drop operations using Rico, you need to first create a draggable object using Rico's Draggable class and then register this object with registerDraggable() function. This is shown here:

```
dndMgr.registerDraggable( new Rico.Draggable('test-rico-dnd','draggableobject') );
```

After registering the Draggable object, you need to register the zone in which the object could be dropped. This is shown here:

```
dndMgr.registerDropZone (new Rico.Dropzone('dropzone1') );
dndMgr.registerDropZone (new Rico.Dropzone('dropzone2') );
```

The code snippet shows that you can drop the draggable object in either dropzone1 or dropzone2.

Let's create an AJAX application using the Rico Framework. The index.html page of Rico application shows a link that forwards you to the rico.jsp page. Listing 31.11 shows the code of index.html page (you can find this file in Code\AJAX\Chapter 31\Rico folder on the CD):

Listing 31.11: The index.html File

```
<html>
<head>
  <link rel="stylesheet" type="text/css" href="style.css">
  <SCRIPT language="javaScript" src="rico.js"></SCRIPT>
  <title>Rico Framework</title>
</head>
<body>
  <h1> Rico Framework </h1>
  <a href="rico.html">Rico</a>
</body>
</html>
```

Figure 31.8 shows the output of index.html page:

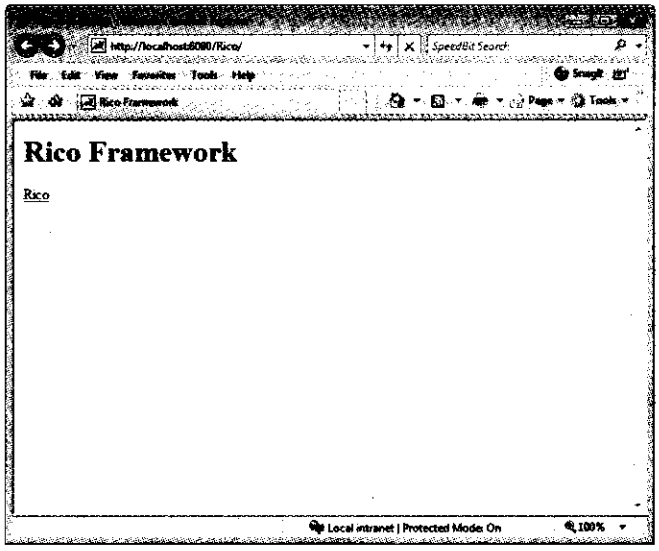


Figure 31.8: Showing Link to Rico Framework

The application starts with a HTML page, rico.html that displays when the user clicks on the Rico hyperlink in Figure 31.8. The page uses the rico.js file, which you can find in the scripts folder of the Rico application. The code, given in listing 31.12, for the rico.html page (you can find this file in Code\AJAX\Chapter 31\Rico folder on the CD):

Listing 31.12: The rico.html File

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta name="generator" content="HTML Tidy, see www.w3.org">
<STYLE type="text/css">
  .borderless {  color:black; text-align:center;  border-width:0; }
  .divclass
  {
        font-family: Ariel,sans-serif;
        font-size: x-small;
        padding: 0px;
        margin: 10 5 5 10;
        height: 100%;
        background-color: white;
        color: red;
  }
</STYLE>
<title>Implementing Rico</title>
<script language="JavaScript" src="scripts/prototype.js"></script>
<script language="JavaScript" src="scripts/rico.js"></script>
<SCRIPT language="JavaScript" type="text/javascript">
window.onload = function ()
{
  ajaxEngine.registerRequest('userpassRequestHandle', 'rico');
  ajaxEngine.registerAjaxElement('userDiv');
  ajaxEngine.registerAjaxElement('passDiv');
  ajaxEngine.registerAjaxElement('message');
}
function retrieveUserPassword(userid)
{
  ajaxEngine.sendRequest( 'userpassRequestHandle',"id="+escape(userid.value));
}
</script>
</head>
<body>
<h3>Enter User Id and it will retrieve Username and Password for the User Using Rico</h3>
<form name="form1" action="signup" method="get">
<table align="left" class="borderless">
<tr>
  <td colspan="3">
  <table align="left" class="borderless">
  <tr>
        <td class="headtext">
        </td>
  </tr>
  </table>
  </td>
</tr>
<tr>
  <td>User ID:</td>
  <td align="left" colspan="2">
<input type="text" id="userid" name="userid"
  onblur="ajaxEngine.sendRequest('userpassRequestHandle','id='+escape(this.value));" >
```

```
  </td>
</tr>
<tr>
  <td>Username:</td>
<td align="left"><div id="userDiv"><input type=text id="inputuser" name="inputuser">
  </div></td>
  <td width="200"><DIV class="divclass" id="message"></div>
</tr>
<tr>
  <td>Password:</td>
<td align="left" colspan="2"><div id="passDiv"><input type=text id="inputpass" size="2"
  name="inputpass"></div></td>
</tr>
</table>
</body>
</html>
```

The page prompts the user to enter the User Id and after that it retrieves the username and password for the entered User Id. The page sends a request to the Servlet, "RicoUserPasswordServlet", which retrieves the username and password for the entered User Id from the database. The code given in Listing 31.13 shows code for the RicoUserPasswordServlet.java file (you can find this file in Code\AJAX\Chapter 31\Rico\src folder on the CD):

Listing 31.13: The RicoUserPasswordServlet.java File

```
package com.kogent.ajax.servlet;
import java.io.IOException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.HashMap;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.kogent.ajax.DatabaseConnector;
import com.kogent.ajax.RicoUtil;

public class RicoUserPasswordServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;

  public void doGet(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException {
      String responseString = null;
      String type = req.getParameter("type");
      if(type==null)
            type = "0";
      String userid = req.getParameter("id");
      if (userid != null) {
            HashMap userpass = retrieveUserPassword(userid);
            if (userpass==null || userpass.size()==0){
                  userpass = new HashMap();
                  userpass.put("userDiv"," ");
                  userpass.put("passDiv"," ");
                  type = "1";
            }
            if (type.equals("1")){
            String message = "User Id: "+userid+" is not in the
            database. Please enter your Username and Password";
            responseString = RicoUtil.buildRicoXML(userpass,"<input>",message);
```

```
                }
                else
                        responseString = RicoUtil.buildRicoXML(userpass,"");
        }
        if (responseString != null) {
                System.out.println(responseString);
                res.setContentType("text/xml");
                res.setHeader("Cache-Control", "no-cache");
                res.getWriter().write(responseString);
        } else {

                res.setContentType("text/xml");
                res.setHeader("Cache-Control", "no-cache");
                res.getWriter().write("?");
        }
    }

    private HashMap retrieveUserPassword(String userid) {
        Connection con = DatabaseConnector.getConnection();
        HashMap userPassMap = new HashMap();
        String queryString = "";
        try {
                queryString = "SELECT username, password FROM userpassword where
                 id="+userid;
                Statement select = con.createStatement();
                ResultSet result = select.executeQuery(queryString);

                while (result.next()) { // process results one row at a time
                        String username;
                        String password;

                        username = result.getString("username");
                        if (result.wasNull()) {
                                username = "";
                        }
                        userPassMap.put("userDiv", username);
                        password = result.getString("password");
                        if (result.wasNull()) {
                                password = "";
                        }
                        userPassMap.put("passDiv", password);
                }
        } catch (Exception e) {
        System.out.println("exception caught getting Username/Password:"
        + queryString + " " + e.getMessage());
        } finally {
                if (con != null) {
                        try {
                                con.close();
                        } catch (SQLException e) {
                        }
                }
        }
        return userPassMap;
    }
}
```

The RicoUserPasswordServlet Servlet connects with the database using the DatabaseConnector.java file. The DatabaseConnector.java file contains JDBC statements for connecting with the database. The source code for this file is same as given in Listing 31.9.

The Servlet, "RicoUserPasswordServlet, retrieves the Username and Password for the entered user Id from the database table named "userpassword" and then builds the Rico data using the RicoUtil.java helper class. This class converts the HashMap data into Rico data. The code given in listing 31.14 shows code for the RicoUtil.java file (you can find this file in Code\AJAX\Chapter 31\Rico\src folder on the CD):

Listing 31.14: The RicoUtil.java File

```
package com.kogent.ajax;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
public class RicoUtil
{
  public static String buildRicoXML(HashMap map, String message)
  {
        StringBuffer ricoXML = new StringBuffer("<ajax-response>");
        String key = "";
        String value = "";
        Iterator it = map.entrySet().iterator();
        while (it.hasNext())
        {
              Map.Entry e = (Map.Entry) it.next();
              value = (String) e.getValue();
              key = (String) e.getKey();
              ricoXML.append("\r\n <response type=\"element\" id=\"" + key
               + "\">" + value + "</response>");
        }
        ricoXML.append("\r\n <response type=\"element\" id=\"message\"
        name=\"message\">"+message+"</response>");
        ricoXML.append("\r\n</ajax-response>");
        return ricoXML.toString();
  }
  public static String buildRicoXML(HashMap map,String element, String message)
  {
        StringBuffer ricoXML = new StringBuffer("<ajax-response>");
        String key = "";
        String value = "";
        Iterator it = map.entrySet().iterator();
        while (it.hasNext())
        {
              Map.Entry e = (Map.Entry) it.next();
              value = (String) e.getValue();
              key = (String) e.getKey();
              ricoXML.append("\r\n <response type=\"element\" id=\"" + key + "\">" +
              "<input type=\"text\" id=\"inner"+key+"\" name=\"inner"+key+"\"
              value=\""+value+"\" /></response>");
        }
        ricoXML.append("\r\n <response type=\"element\" id=\"message\"
        name=\"message\">"+message+"</response>");
        ricoXML.append("\r\n</ajax-response>");
        return ricoXML.toString();
  }
  public static String buildRicoObjectXML(HashMap map, String message)
  {
        StringBuffer ricoXML = new StringBuffer("<ajax-response>\r\n <response
        type=\"object\" id=\"locationUpdater\"><location ");
        String key = "";
```

```
            String value = "";
            Iterator it = map.entrySet().iterator();
            while (it.hasNext())
            {
                    Map.Entry e = (Map.Entry) it.next();
                    value = (String) e.getValue();
                    key = (String) e.getKey();
                    ricoXML.append(key+"=\""+ value +"\" " );
            }
            ricoXML.append("/></response>\r\n</ajax-response>");
            return ricoXML.toString();
    }
}
```

The view created by the Rico.html page is same as the one shown in Figure 31.9 When you enter the User Id 1; it displays the username and password of the user from the database, as shown in Figure 31.9:

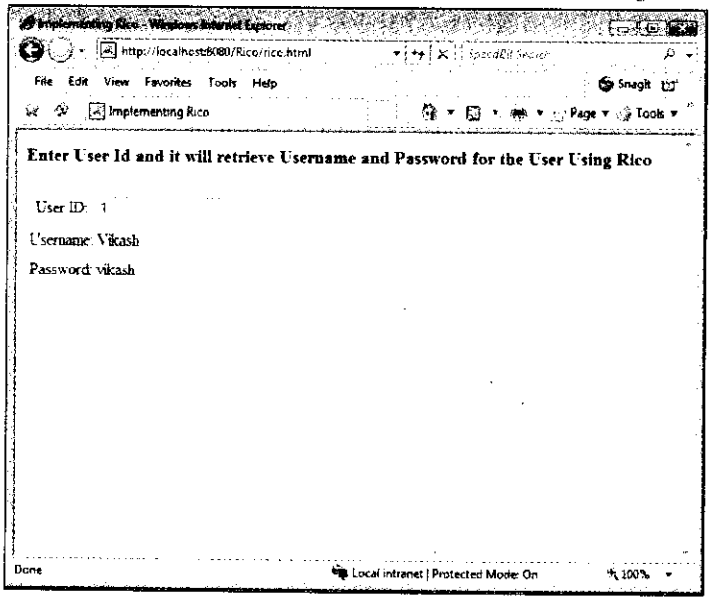


Figure 31.9: Retrieving Username and Password using Rico

In this section, you have learned using the Rico Framework to create an AJAX application. Let's discuss the Spry Framework in detail, in the following section.

Working with Spry Framework

The Spry Framework for AJAX, which is created by Adobe, is a JavaScript library for Web designers. It is free for use. The Spry Framework provides the functionality to Web designers to build web pages which have AJAX functionality. The Spry Framework is not yet supported by other frameworks. It is heavily based on HTML and it is developed for the users with the basic idea of HTML, CSS, and JavaScript. The HTML used in the framework is very simple and the use of JavaScript is minimal. With Spry, you can simply fill your HTML with dynamic data.

The key advantage of Spry Framework is that it is HTML centric and so the Web designers can use their existing knowledge to design rich user interfaces with AJAX features and there is no need to learn any new language or programming model. The web page design by using Spry is easy to understand, as it is written in HTML and JavaScript. The user can view the source of the web page and easily understand.

The Spry Framework includes the use of the Google XPath library. As a result, Spry may break when used in conjunction with the Prototype Ajax library or other JavaScript libraries. The Prototype Ajax library extends the JavaScript built-in types and so it breaks the Google XPath implementation, and other third-party JavaScript libraries. The creators of Spry have taken steps to ensure compatibility with Prototype and other frameworks. While working with Spry, you need to enable JavaScript support in your browsers. The browsers supported by Spry Framework are as follows:

- FireFox 1.5 +(Windows and Mac)
- Netscape 7.2 (Windows)
- Internet Explorer 6+ (Windows)
- Safari 2.0.3 (Mac)

The Spry Framework is a way to incorporate XML, JSON, or HTML data into pages using HTML, CSS, and a minimal amount of JavaScript, without the need for refreshing the entire page. Spry also provides easy to build and style widgets, providing advanced page elements for end users. The important thing related with Spry is that it uses XML, JSON, and HTML datasets. You cannot use XML files for datasets. The web page can be written in any scripting language, like JavaScript, ColdFusion, PHP, and ASP.

Let's create an AJAX application using the Spry Framework. The application validates the date entered by the user. The Spry Framework comes with many widgets that provide JavaScript libraries for validation of various form controls. In this application, we are going to use these in-built widgets. The framework also provides CSS files for applying stylesheet onto the form. All these widgets and CSS files come with the downloaded zip. The application has only one page index.html, which uses these widgets and CSS files. The code given in Listing 31.15 shows code for the index.html file (you can find this file in the Code/AJAX/Chapter 31/SpryValidation folder on the CD):

Listing 31.15: The index.html File

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Form Validation Using Spry Widgets</title>
<link href="widgets/textfieldvalidation/SpryValidationTextField.css" rel="stylesheet"
  type="text/css" />
<link href="widgets/selectvalidation/SpryValidationSelect.css" rel="stylesheet"
  type="text/css" />
<link href="widgets/textareavalidation/SpryValidationTextarea.css" rel="stylesheet"
  type="text/css" />
<link href="widgets/checkboxvalidation/SpryValidationCheckbox.css" rel="stylesheet"
  type="text/css" />
<link href="widgets/radiovalidation/SpryValidationRadio.css" rel="stylesheet"
  type="text/css" />
<link href="validation.css" rel="stylesheet" type="text/css" media="all" />
<script type="text/javascript"
  src="widgets/textfieldvalidation/SpryValidationTextField.js"></script>
<script type="text/javascript"
  src="widgets/selectvalidation/SpryValidationSelect.js"></script>
<script type="text/javascript"
  src="widgets/textareavalidation/SpryValidationTextarea.js"></script>
<script type="text/javascript"
  src="widgets/checkboxvalidation/SpryValidationCheckbox.js"></script>
<script type="text/javascript"
  src="widgets/radiovalidation/SpryValidationRadio.js"></script>
</head>
<body>
<noscript><h1>This page requires JavaScript. Please enable JavaScript in your browser and
  reload this page.</h1></noscript>
<div id="wrap">
<h1>Validation using Spry Framework</h1>
```

```
<div id="centralColumn">
  <div id="articles">
    <form id="form1" name="form1" method="get" action="index.html">
    <div id="formregion">

    <div id="theUsername">
    <div class="formLabel">Name:</div>
    <input name="yourName" type="text" id="yourName"  />
    <img src="images/ok.gif" title="Valid" alt="Valid" class="validMsg" border="0"/>
    <span class="textfieldRequiredMsg">Please enter your Name.</span>
    </div>
    <div id="theDate">
    <div class="formLabel">Date of Birth:</div>
    <input name="birthDate" type="text" id="birthYear" value="" />
    <img src="images/ok.gif" title="Valid" alt="Valid" class="validMsg" border="0"/>
    <span class="textfieldRequiredMsg">Please enter a date.</span>
    <span class="textfieldInvalidFormatMsg">Please enter a valid date
    (mm/dd/yyyy).</span>
    </div>
    <div id="radios">
        <div class="formLabel">Maritial Status:</div>
        <label for="yes_married">Yes
        <input type="radio" name="maritialstatus" id="yes_married"
        value="Y"/></label>
        <label for="no_married">No
        <input type="radio" name="maritialstatus" id="no_married"
        value="N"/></label>
        <img src="images/ok.gif" title="Valid" alt="Valid" class="validMsg"
        border="0"/>
        <span class="radioRequiredMsg">Please specify a value.</span>
    </div>
    <div id="theAddress">
        <div class="formLabel">Address:</div>
        <table border="0">
        <tr>
            <td valign="top" rowspan="2">
            <textarea name="yourAddress" id="yourAddress" cols="45"
            rows="5"></textarea>
            </td>
            <td valign="top" width="325">
            <img src="images/ok.gif" title="Valid" alt="valid"
            class="validMsg" border="0"/>
            <span class="textareaRequiredMsg">Please enter your
            Address.</span>
            <span class="textareaMinCharsMsg">Please enter at least 20
            characters.</span>
            </td>
        </tr>
        <tr>
        <td valign="bottom"><span id="countvalidta1"> </span> / 180</td>
        </tr>
        </table>
    </div>
    <div id="theRole">
        <div class="formLabel">Your Role:</div>
        <select name="yourRole" id="yourRole">
        <option value="" selected="selected">Please select</option>
        <option value="Java Programmer">Java Programmer</option>
        <option value="Senior Java Programmer">Senior Java Programmer</option>
        <option value="HR Manager">HR Manager</option>
```

```
            <option value="Team Leader">Team Leader</option>
            <option value="Project Leader">Project Leader</option>
            <option value="Project Manager">Project Manager</option>
            </select>
            <img src="images/ok.gif" title="Valid" alt="Valid" class="validMsg"
            border="0"/>
            <span class="selectRequiredMsg">Please select a Role.</span>
    </div>
    <div id="checkboxes">
            <div class="formLabel">Skills:</div>
            <div class="validcheckbox">
            <label for="checkbox_1">Java SE</label>
            <div class="input_container">
            <input type="checkbox" name="yourSkill" id="checkbox_1" value="1"/>
            </div>
            </div>
            <div class="validcheckbox">
            <label for="checkbox_2">Java EE</label>
            <div class="input_container">
            <input type="checkbox" name="yourSkill" id="checkbox_2" value="2"/>
            </div>
            </div>
            <div class="validcheckbox">
            <label for="checkbox_3">Struts</label>
            <div class="input_container">
            <input type="checkbox" name="yourSkill" id="checkbox_3" value="3"/>
            </div>
            </div>
            <div class="validcheckbox">
            <label for="checkbox_4">Spring</label>
            <div class="input_container">
            <input type="checkbox" name="yourSkill" id="checkbox_4" value="4"/>
                    </div>
            </div>

            <div class="validcheckbox">
            <label for="checkbox_5">JSP/JSF</label>
            <div class="input_container">
            <input type="checkbox" name="yourSkill" id="checkbox_5" value="5"/>
            </div>
            </div>
            <div id="errors">
            <img src="images/ok.gif" title="Valid" alt="Valid" class="validMsg"
            border="0"/>
            <span class="checkboxRequiredMsg">Please select at least one
            skill.</span>
            <span class="checkboxMaxSelectionsMsg">Only a maximum of 3 skills can be
            selected.</span>
            </div>
            <br style="clear:both"/>
    </div>
    <div id="theRateskill">
            <div class="formLabel">Rate your skills level:</div>
            <select name="Rating" id="skillsRating">
                    <option value="">Please select</option>
                    <option value="5">*****</option>
                    <option value="4">****</option>
                    <option value="3">***</option>
                    <option value="2">**</option>
```

```
                        <option value="1">*</option>
                    </select>
                    <img src="images/ok.gif" title="Valid" alt="Valid" class="validMsg"
                    border="0"/>
                    <span class="selectRequiredMsg">Please Rate your skills.</span>
            </div>
            <div class="buttons">
            <input type="submit" name="Submit" id="submit" value="Submit"/>
            <input type="reset" name="Reset" id="reset" value="Reset" />
            </div>
    </div>
    </form>
</div>
<div class="clearAll"> </div>
</div>
</div>
<script type="text/javascript">
<!--
var theUsername = new Spry.Widget.ValidationTextField("theUsername", "none",
    {useCharacterMasking:true, validateOn:["change"]});
var theDate = new Spry.Widget.ValidationTextField("theDate", "date",
    {useCharacterMasking:true, format:"mm/dd/yyyy", hint:"mm/dd/yyyy",
    validateOn:["change"]});
var theRateskill = new Spry.Widget.ValidationSelect("theRateskill",
    {validateOn:["change"]});
var theRole = new Spry.Widget.ValidationSelect("theRole", {validateOn:["change"]});
var theAddress = new Spry.Widget.ValidationTextarea("theAddress",
    {useCharacterMasking:true, minChars:20, maxChars:180, counterType:"chars_count",
    counterId:"Countvalidta1", validateOn:["change"]});
var checkboxes = new Spry.Widget.ValidationCheckbox("checkboxes", {validateOn:["change"],
    maxSelections:3});
var dvdVersion = new Spry.Widget.ValidationRadio("radios", {validateOn:["change"]});
//-->
</script>
</body>
</html>
```

In Listing 31.15, many JavaScript files are included. These files are provided by the Spry Framework. You can use the files in your application by including the following lines in your source files:

```
<script type="text/javascript"
    src="widgets/textfieldvalidation/SpryValidationTextField.js"></script>
<script type="text/javascript"
    src="widgets/selectvalidation/SpryValidationSelect.js"></script>
<script type="text/javascript"
    src="widgets/textareavalidation/SpryValidationTextarea.js"></script>
<script type="text/javascript"
    src="widgets/checkboxvalidation/SpryValidationCheckbox.js"></script>
<script type="text/javascript"
    src="widgets/radiovalidation/SpryValidationRadio.js"></script>
```

Listing 31.15 also uses various CSS files for various form controls. These files are included using the following lines:

```
<link href="widgets/textfieldvalidation/SpryValidationTextField.css" rel="stylesheet"
    type="text/css" />
<link href="widgets/selectvalidation/SpryValidationSelect.css" rel="stylesheet"
    type="text/css" />
<link href="widgets/textareavalidation/SpryValidationTextarea.css" rel="stylesheet"
    type="text/css" />
<link href="widgets/checkboxvalidation/SpryValidationCheckbox.css" rel="stylesheet"
    type="text/css" />
<link href="widgets/radiovalidation/SpryValidationRadio.css" rel="stylesheet"
    type="text/css" />
<link href="validation.css" rel="stylesheet" type="text/css" media="all" />
```

NOTE

The CSS and Validation files are provided in the Code/AJAX/Chapter 31/SpryValidation folder on the CD.

After including all these files, we next create various input controls on the page. These controls are checkboxes, radio buttons, text fields, textarea, and select box. After creating these fields, we pass the id of these fields to the corresponding validation class defined in Spry.Widget.* class, e.g. to provide validation on a textfield, we use the Spry.Widget.ValidationTextField class. This can be done as shown here:

```
var theUsername = new Spry.Widget.ValidationTextField("theUsername", "none",
  {useCharacterMasking:true,  validateOn:["change"]});
var theDate = new Spry.Widget.ValidationTextField("theDate", "date",
  {useCharacterMasking:true, format:"mm/dd/yyyy", hint:"mm/dd/yyyy",
  validateOn:["change"]});
var theRateskill = new Spry.Widget.ValidationSelect("theRateskill",
  {validateOn:["change"]});
var theRole = new Spry.Widget.ValidationSelect("theRole", {validateOn:["change"]});
var theAddress = new Spry.Widget.ValidationTextarea("theAddress",
  {useCharacterMasking:true, minChars:20, maxChars:180, counterType:"chars_count",
  counterId:"Countvalidtal", validateOn:["change"]});
var checkboxes = new Spry.Widget.ValidationCheckbox("checkboxes", {validateOn:["change"],
  maxSelections:3});
var dvdVersion = new Spry.Widget.ValidationRadio("radios", {validateOn:["change"]});
```

All these classes have the validationOn parameter, which specifies that the validation will be performed when the value of the field will be changed. When you run the application on the server, it displays a screen similar to the one shown in Figure 31.10:

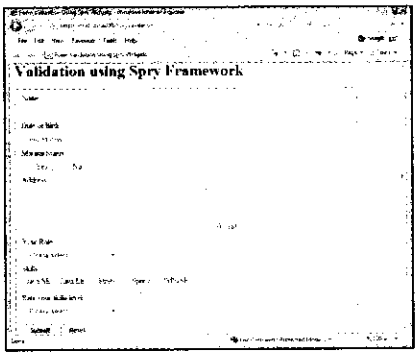


Figure 31.10: Validation using Spry Framework

Suppose a user doesn't fill the form properly and leaves some field or enters some values which are not valid for that field. In this case, an appropriate error message will be displayed in front of that field, as shown in Figure 31.11:

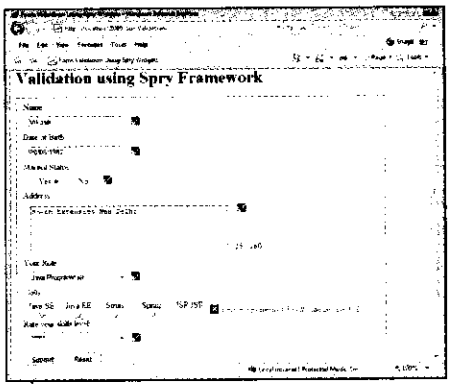


Figure 31.11: Showing Validation Messages

In Figure 31.11, you can see that the user enters values in Name, Date of Birth, Marital Status, Address, Your Role, and Rate your skills level fields, according to the desired format and therefore a right sign is displayed in front of these fields. When the user fails to fill the Skills field in the desired format, the field is marked with a cross sign.

In this section, you have learned using the Spry Framework to create AJAX application. Let's quickly summarize some other commonly used AJAX frameworks in the following section.

Commonly used frameworks for AJAX

All the frameworks, except Prototype, Script.aculo.us and DWR discussed in the preceding sections, are not widely used. In this section, we will discuss three widely used AJAX Frameworks, which are Google Web toolkit, Yahoo UI library, and AJAX.NET. We will only give you a brief introduction of these frameworks.

Yahoo UI Library

The Yahoo UI library provides a set of utilities and controls, which are used for developing rich internet applications. These utilities and controls are written in JavaScript. The Yahoo UI library uses techniques, such as DOM scripting, DHTM, CSS, and AJAX. The Yahoo UI library is available as open source and is free for all uses. `Yahoo UI library` was developed by the Yahoo's client-side platform engineering team. The project leader was Thomas Sha. Some of the components of the `Yahoo UI library` works in the background and makes the browser programming simpler. Examples of such components include Event Utility. Components, such as `calendar` controls, are examples of visual widgets. These widgets have default look-and-feel.

Google Web Toolkit

Generally, `AJAX` applications are created by using JavaScript libraries with XHTML and CSS. In this chapter we discussed many AJAX Frameworks and libraries by which developers can use pre-defined `JavaScript` classes to implement features, like drag-and-drop, tree structures, validation, etc. These libraries are Dojo toolkit, Prototype, Rico, `Script.aculo.us, Jquery, and Spry`. These libraries are for those developers who are already advanced in their `JavaScript` knowledge. If you don't have advance knowledge of JavaScript then you will feel uncomfortable while using these libraries.

Here, the need of Google Web Toolkit (GWT) arises. GWT is a framework, which is used for creating rich and redistributable user-interface `widgets`. The difference between `GWT` and other `AJAX` framework is that with `GWT` you write your code in Java, instead of `JavaScript`. The `GWT` approach for developing `AJAX` applications is different from other toolkits. GWT uses Java-to-JavaScript compiler for converting your Java code into JavaScript. The user interface is also created in Java language and it compiles into HTML and JavaScript.

AJAX.NET Professional

AJAX.NET Professional (available at `www.ajaxpro.info`) is an AJAX Framework designed for use with the .NET Framework, specifically ASP.NET. As with JPSpan and DWR, AJAX.NET allows developers to create classes on the server that can be accessed via the client. This is done once again by abstracting the communication between the client and server, and providing cross-browser communication behind the scenes. This framework uses the .NET version of reflection to inspect the classes and create a JavaScript interface. AJAX.NET Professional also takes advantage of a .NET feature called attributes, which are used to mark specific methods that should be available to the client. This allows developers to concentrate on the true object-oriented design without worrying about exposing sensitive methods to the client.

Following are the features of AJAX .Net Professional framework:

AJAX library for ASP.NET provides the very basic AJAX capability to make `xmlhttp` callbacks. It does not have any 'AJAX-enabled' controls or support for Viewstate, etc.

- It can serialize any data type from .NET including custom classes, structures or enum types.
- It can include JavaScript DataTable, DataSet, and support for IList and IEnumeration, XML documents.
- It supports Internet Explorer 5.0 and higher, FireFox, Opera, Safari, and Netscape.
- It is a great ActiveX replacement for Internet Explorer, if ActiveX are disabled.
- It uses JSON instead of XML for performance reason.

- It reduces HTML traffic (only data is sent instead of HTML code).

We will discuss more about ASP.NET AJAX in Chapter 32.

Summary

In this chapter, we discussed the various AJAX Frameworks and how to use some of these frameworks in your application. We covered some of the widely used Ajax frameworks, which are available in the market. As you can see, there is a considerable range of server-side frameworks, and the one you select depends upon the language you use on your server (PHP, Java, .NET, and so on). For each language, there are multiple frameworks available.

In the next chapter, we will discuss Developing ASP.NET AJAX Application.

Quick Revise

Q1. List the AJAX frameworks used for Web application development.

Ans: The commonly used AJAX frameworks are as follows:

- Jquery Framework
- Prototype Framework
- Script.aculo.us Framework
- DOJO Toolkit Framework
- DWR Framework
- JPSpan Framework
- Rico Framework
- Spry Framework

Q2. Define the Jquery Framework.

Ans: Jquery was created by John Resig in early 2006. It is a great library for persons who are using JavaScript code and is beneficial for both a beginner and an experienced programmer in JavaScript. Jquery enables developer to code easily. It does not require repeating loops and DOM scripting library calls. Jquery uses least characters to express anything. Thus, the philosophy of Jquery is unique. It is designed to keep the things simple and reusable.

Q3. Define the Prototype Framework.

Ans: The Prototype Framework is a JavaScript Framework, which provides an AJAX Framework and other utilities. Ruby on Rails integrates the Prototype Framework. Prototype provides various functions for developing JavaScript applications. The features range from programming shortcuts to major functions for dealing with `XMLHttpRequest`.

Q4. Define the Dojo Framework.

Ans: Dojo is an Open Source DHTML toolkit written in JavaScript. It builds on several contributed code bases (nWidgets, f(m) and Burstlib), which is why we refer to it sometimes as a "unified" toolkit. Dojo aims to solve some long-standing historical problems with DHTML which prevented mass adoption of dynamic Web application development.

Q5. Define the DWR Framework.

Ans: Direct Web Remoting (DWR) is an AJAX Framework for Java/JSP. DWR is an excellent way to harness the power of Java for use in Web applications. Joe Walker, the primary developer of DWR, constantly updates the framework with new functionality designed to make Web applications more responsive. Since DWR works with literally hundreds of available Java packages, chances are that there's something already available that can perform the operations necessary for your application. There is also the advantage that there is no shortage of information, tutorials, and experts willing to share their knowledge about the Java platform.

Q6. Define the JPSpan Framework.

Ans: JPSpan is an AJAX Framework built with the intention of integrating client-side JavaScript with server-side code written in PHP. JPSpan works by analyzing a PHP class by using reflection. It then emits a number of JavaScript methods that accepts the same arguments as the methods of the class on the server. When one of these methods is called, it uses a cross-browser library to instantiate the appropriate XMLHttpRequest for the platform it is running on, and posts the data to the server. The response is then read and passed to a function on the client for further use within the page.

Q.7 Define Rico Framework.

Ans: Rico is an AJAX Framework, which is based on JavaScript and provides the functionality of drag-and-drop. It is based on Prototype Framework, which is already discussed in this chapter. Rico is provided open source for either personal or commercial use. You can use Rico for building Accordion component, which are found in Macromedia Flex and Laszlo. You can design a very rich interface by using Rico's style.

Q8. Define Spry Framework.

Ans: The Spry Framework for AJAX, which is created by Adobe, is a JavaScript library for Web designers. It is free for use. The Spry Framework provides the functionality to Web designers to build web pages which have AJAX functionality. The Spry Framework is not yet supported by other frameworks. It is heavily based on HTML and it is developed for the users with the basic idea of HTML, CSS, and JavaScript. The HTML used in the framework is very simple and the use of JavaScript is minimal. With Spry, you can simply fill your HTML with dynamic data.

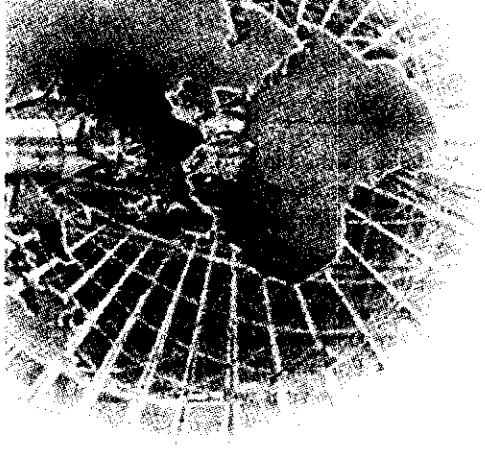
Q9. List the browsers supported by the SpryFramework

Ans: The browsers supported by Spry Framework are as follows:

- FireFox 1.5 +(Windows and Mac)
- Netscape 7.2 (Windows)
- Internet Explorer 6+ (Windows)
- Safari 2.0.3 (Mac)

Q10. Define GWT.

Ans: Google Web Toolkit (GWT) is a framework, which is used for creating rich and redistributable user-interface `widgets`. The difference between `GWT` and other `AJAX` framework is that with `GWT` you write your code in Java, instead of `JavaScript`. The `GWT` approach for developing `AJAX` applications is different from other toolkits. GWT uses Java-to-JavaScript compiler for converting your Java code into JavaScript. The user interface is also created in Java language and it compiles into HTML and JavaScript



32

Developing ASP.NET AJAX Applications

You are already familiar with the Web applications that use the `postback` model or method to refresh the page contents. In the `postback` model, with every request made by an application viewer, the entire page is sent back to the server. The server then processes the request and performs modifications on the page, and sends the regenerated page to the viewer's browser to display the requested information.

This `postback` model has been in use since the beginning of the Internet. The advent of advance programming systems, such as ASP.NET, has tried to make user's Web experience more responsive and interactive due to the ability to perform complex tasks and update the Web pages swiftly. However, in comparison to the window applications, the Web applications are still less interactive, responsive, and seamless. Fortunately, today, there is no compulsion to use the `postback` model in the websites. Now, it is possible to program your websites in such a way that it can update the page information faster, without page flicker and without interrupting user interactions.

In the real world scenario, we can take example of Google E-mail Service, i.e. Gmail. In Gmail, when we need to attach a document with the text mail, we simply browse and attach the required document, while continuing to write the text message. The process of attaching the document is performed in the background. In other words, we can say that the user is not compelled to wait until the document is completely attached with the mail. The processing of attaching the document in background is possible with a new technology, called `AJAX`. In `AJAX`, a call to server for page refreshment takes place in the background and the user does not even get a hint of when the request is sent to the server and new data is retrieved from the server. The user just gets the new content or requested information. The call that takes place in the background is called `Asynchronous call`. Another major feature of `AJAX` is `partial updation`, in which it updates only a particular portion of a Web page without touching the rest.

In this chapter, we first learn what AJAX is and how is it different from other technologies. We then explore ASP.NET AJAX, its components, and ASP.NET technology that uses the `AJAX` approach for Web development. We also discuss the extension controls delivered with Visual Studio 2008 and their use to implement `AJAX` functionality in your Web application. In the end, we learn about the ASP.NET AJAX Control Toolkit (ACT) and its controls.

AJAX—A New Approach

AJAX is neither a new programming language nor a new platform for developing websites. We can call `AJAX` a new approach, because it displays the refreshed contents on a Web page by using the Page Update approach rather than the Page Replacement approach, which we used in our traditional websites. `AJAX` is built on existing technologies, such as DOM, CSS, XHTML, and the rest. `AJAX` is a collection of development components, tools, and techniques for creating highly interactive Web applications. `AJAX` uses client scripts to make asynchronous calls to a server and loads only those parts of a Web page that needs to be changed. The transfer of data from the server is carried out over the `XMLHTTP` protocol in the form of packets by using XML and JSON.

The following are the technologies involved in building `AJAX`:

- JavaScript
- XHTML
- Cascading Style Sheet (CSS)
- Document Object Model (DOM)
- XMLHttpRequest Object

Exploring JavaScript

`JavaScript` is a client-side dynamic scripting language that supports object-oriented programming. `JavaScript` provides tools to communicate with the back-end server. There many other client-side scripting languages but `AJAX` includes `JavaScript`, because `JavaScript` is the only client-side scripting environment supported by all modern Web browsers. `JavaScript` is commonly hosted in a browser to add interactivity to HTML pages. `JavaScript` is an interpreted language, not a compiled language. It does not require static type checking as in C++ and C#. In addition, you can declare a variable without specifying its type in `JavaScript`. This makes it easier for the users to work with JavaScript.

Introducing XHTML

XHTML is a combination of HTML and XML. The objective behind XHTML is to combine best features of XML and HTML. It is quite similar to HTML 4.0 and contains a set of tags, similar to HTML. However, by using XHTML, developers can define their own tags and attributes when the existing set of tags does not meet their requirements.

XHTML is used with AJAX-enabled website to ensure that the website run consistently on all browsers, including desktop browsers (IE, Mozilla Firefox, Opera) and browsers for the hand-held devices (WAP browser). Using XHTML is preferred over using HTML because sometimes we came across Web pages containing inappropriate HTML tags, such as an opening tag does not contain a closing tag. The Web pages containing inappropriate HTML tags can run on desktop browsers but not on the browsers of hand-held devices. XHTML resolves these issues by generating errors while running such applications and ensures that the applications are compatible with all the browsers. In addition, XHTML is based on XML; which implies that it is a platform-independent language.

Introducing CSS

CSS is a file that defines the display pattern of a Web Form. Through CSS, we can specify fonts, colors, styles (bold, italic), size, and the rest. It is a text document that can be created by using any text editor that has the `.css` extension. After defining presentation styles in the CSS file, you need to attach this CSS file with the Web Form. The following code snippet shows an example to define how the body of the document is manipulated using the CSS:

```
BODY
{
  Arial, Helvetica, sans-serif;
  Color:            black;
  Background-color: white;
}
```

You can manipulate the design of your Web Form by changing the properties of each control too; however, AJAX uses CSS for the following reasons:

- Since CSS is a separate file, you can attach a reference of a single CSS file with more than one Web forms. This is beneficial as you do not need to define similar type of formatting styles again and again.
- As CSS is a separate file, it separates the working of form definer and designer, which in turn makes it easy for other developers to understand both the definition and design system of a Web Form clearly.
- CSS is cached in the browser memory in the first visit to the website, which helps in the processing of further requests on the same website quickly.
- CSS also enables you to represent same content in different ways by using different style sheets.

Exploring DOM

DOM represents a structured Web form in an object-oriented model. DOM is a platform- and language-independent interface that allows programs and scripts to access and update the contents of a Web form. DOM is recommended by W3C to manipulate page contents at runtime.

DOM takes the path of a Web form as a parameter and represents it in a tree structure (called document tree) by defining every element as a node in its inner memory. It maintains relationship between the nodes of the document tree, such as parent node, child node, and the rest. This can be understood with the help of Figure 32.1:

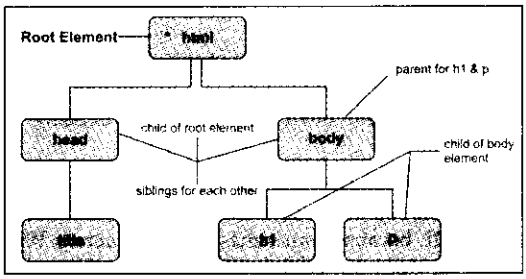


Figure 32.1: The Tree Representation of HTML Document

The preceding figure shows various nodes that can be a part of a Web Form, and the relationship between the nodes.

DOM has a rich set of methods (such as `insertBefore`, `insertAfter`, `hasChildNodes`) that are used to access and change these nodes and their corresponding values. The changes made to the nodes and their corresponding values are incorporated back to the Web Form to reflect updated content on the Web form. AJAX, which is also a platform- and language-independent technology, uses DOM to update the contents of Web page, whenever required.

Introducing XMLHttpRequest Object

The core of AJAX functionality is the `XMLHttpRequest` object, as it is responsible for making asynchronous requests to the server, and updating the server response and performing necessary changes in the current Web page. We can say that this object is the core of ASP.NET AJAX. It receives and sends data in the form of XML. It contains various members in form of properties, methods, and events. Noteworthy properties of the `XMLHttpRequest` object are listed in Table 32.1:

Table 32.1: Noteworthy Properties of the XMLHttpRequest Object

Property	Description
`onreadystatechange`	Obtains or sets the event handler for asynchronous requests
`readyState`	Obtains the current state of the request operation
`responseBody`	Obtains the response body as an array of unsigned bytes
`responseText`	Obtains the response body as a string
`responseXML`	Obtains the response body as an XML Document Object Model (DOM) object
`Status`	Obtains the HTTP status code of the request
`statusText`	Obtains the friendly HTTP status of the request

Noteworthy methods of the `XMLHttpRequest` object are listed in Table 32.2:

Table 32.2: Methods of the XMLHttpRequest Object

Method	Description
`Abort`	Aborts the current HTTP request
`getAllResponseHeaders`	Sends back the complete list of response headers
`getResponseHeader`	Sends back the specified response header
`Open`	Allocates method, destination URL, and other optional attributes of a pending request
`Send`	Conveys an HTTP request to the server and retrieves a response

Table 32.2: Methods of the XMLHttpRequest Object

Method	Description
setRequestHeader	Inserts custom HTTP headers to the request

NOTE

XMLHttpRequest object has an additional property named Timeout, with IE 8. The Timeout property obtains and sets the time out value in milliseconds for which client has to wait for receiving response from server. IE 8 also provide an event named Ontimeout to XMLHttpRequest object to instruct browser what to do when time-out period expires.

Need for AJAX

AJAX has established a milestone in the history of Web development. Considering developers' perspective, AJAX has given them a new approach to design and implement functionality in their Web applications. They are not bound to follow the old and somewhat tedious request-response model. AJAX has saved their labor from designing more Web pages for displaying new information to the user. This is possible since AJAX reloads only the required section of a Web page, unlike other technologies. With the introduction of AJAX, the scalability of traditional applications has improved.

Considering users' perspective, AJAX has made their Web interaction faster, responsive, and more importantly stress-free. This is money-saving too, as they are not bound to buy new browsers to allow AJAX to enable the websites run properly, because it is compatible with most of the current browsers in use, such as IE6 and advanced versions, Firefox 1.0 and advanced versions, and the rest. AJAX-enabled websites can work easily even with slow connections too, such as dial-up connection.

ASP.NET AJAX fulfills its aim of providing a comprehensive Web development environment that is rich with client-scripting features, and supports ASP.NET server-side development. Comparing with the previous versions, ASP.NET 3.5 has in-built AJAX's script libraries and server components to meet the complexity in developing AJAX-enabled websites.

AJAX and Other Technologies

How does AJAX differ from other technologies, such as ASP.NET, JSP, which use the postback model? If this is not a new technology, as mentioned earlier, then what makes AJAX so popular? To answer these questions in an understanding manner, we first need to analyze how other technologies work. Figure 32.2 shows the traditional Web application model:

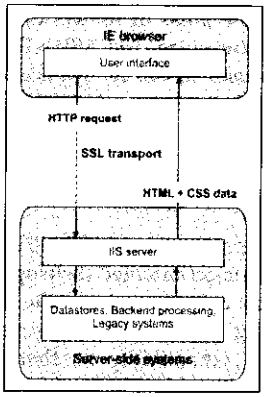


Figure 32.2: The Traditional Web Application Model

Traditional Web applications are based on the request-response or postback model. For each client interaction with the UI of a traditional Web application, an HTTP request is sent to the server, which processes the request and sends the results back to the application. At the time, when this request-response process is in progress,

seeing a blank screen with an hourglass mouse pointer can be quite frustrating at times. Considering today's modern scenario, when more and more transactions are taking place through the Internet from all over the world, this time-consuming process can be frustrating.

To remove this shortcoming, AJAX, which implements an entirely different approach to Web development, has been introduced. Figure 32.3 shows the AJAX Web application model:

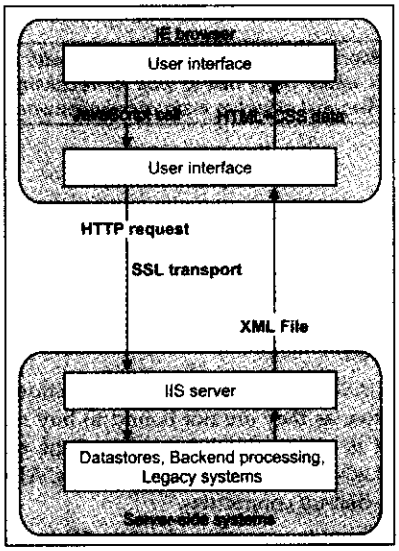


Figure 32.3: The AJAX Web Application Model

The AJAX Web application model handles the client-side activities, such as validations without making a round trip to the server. The tasks that are required to be processed at the server-end are handled by the AJAX engine. The AJAX engine makes asynchronous calls to the server without interrupting the user's interactions with the UI; therefore, it enhances the user-interactivity and performance, which makes AJAX different from other technologies.

ASP.NET AJAX Architecture

Every technology has its working domain. This domain is defined by its architecture, which specifies how that specific technology works and what are the areas of its working. ASP.NET AJAX architecture comprises the client-side architecture and server-side architecture. The ASP.NET AJAX client-side architecture is nothing, but the client scripts that a Web application uses to call an application or a service on a server. ASP.NET AJAX server-side architecture, on the other hand, comprises ASP.NET Web services, ASP.NET server controls, and ASP.NET AJAX Extension controls (specific to implement AJAX functionality in website). Now, let's explore both client-side architecture and server-side architecture one by one.

Client-Side Architecture

The ASP.NET AJAX client-side architecture comprises ASP.NET AJAX client script libraries, which contain a number of `JavaScript` files that support object-oriented development so that the client browser can request the server asynchronously. This asynchronous call support is quite new in a scripting environment and instills a high level of modularity in client scripting. Client-side architecture contains non-visual components, controls, and behavior classes for cross-browser compatibility so that ASP.NET AJAX-enabled website can work with Web browsers, such as Internet Explorer, Mozilla Firefox, and Opera. Moreover, client-side Architecture shares data with Web servers mostly in the form of JSON and XML format, because both these formats are platform-independent and easy to create and parse for client and server. Figure 32.4 shows the client-side architecture:

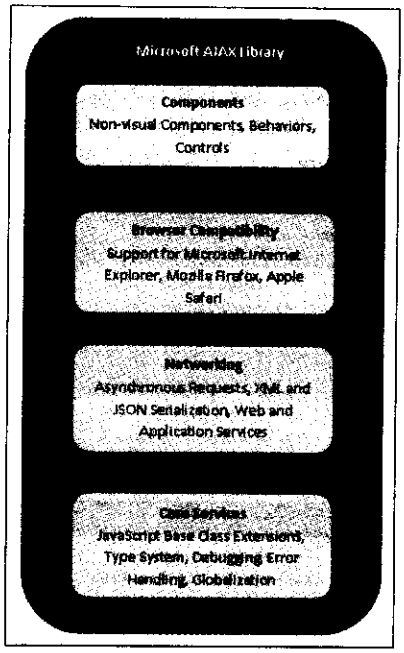


Figure 32.4: The Client-side Architecture

The client-side architecture comprises a number of layers, which are as follows:

- **Components**—The client components, such as `Timer` control are used to get robust functionalities without performing `postbacks`. The use of components varies as per the type of client behavior needs to be displayed. There are three categories of components:
  - **Components**—Objects encapsulating the code for performing the task, but these are not visible in the browser, such as timer object.
  - **Behaviors**—Enhance the basic behavior of the existing `DOM` objects.
  - **Controls**—Create new `DOM` objects having custom behaviors.
- **Browser Compatibility**—With the browser compatibility support, the AJAX library supports the Internet Explorer, Mozilla Firefox, and Apple Safari. Therefore, a single script needs to be written for all the browsers instead of varied scripts for different browsers.
- **Networking**—Networking support deals with the communication between script in the browser and Web services and applications. Moreover, it manages asynchronous remote method calls, which results in usage of the networking layer automatically, without providing any code. It also supports accessing of server-based form authentication, role information, and profile information in the client script.
- **Core Services**—`JavaScript` files used in the client-script libraries provide support for the object-oriented development. This results in a higher degree of consistency and modularity in client scripting. The AJAX client-side library offers the following core services:
  - Object-oriented extensions to `JavaScript`, such as classes, namespaces, inheritance, and so on.
  - A base class library, which contains components like string builders, and extended error handling.
  - Support for JavaScript libraries that are either embedded in an assembly or provided on the hard-disk drives.

In addition, the core services also include the `Sys.Debug` class and error objects to support debugging and error handling. The `Sys.Debug` class provides methods for displaying objects in the readable form at the end of the Web page, and shows trace messages, which enable the usage of assertions. One another service, globalization, enables the usage of a single code base to provide user interface for multiple locales.

Server-Side Architecture

ASP.NET AJAX server-side architecture contains a set of server controls and components for organizing user interface and flow of application's execution, and Web services for authenticating and defining roles, and managing user profiles. It also manages serialization, validation, control extensibility, and the rest.

Figure 32.5 shows the server-side architecture:

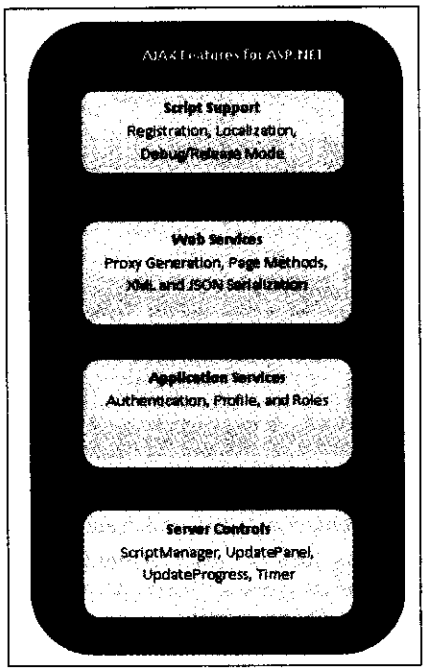


Figure 32.5: The Server-side Architecture

The various parts of the server-side architecture are:

- Script Support
- Web Services
- Application Services
- Server Controls

Now, let's discuss these parts of the server-side architecture in detail one by one.

Script Support

Server-side architecture of ASP.NET contains a rich set of `JavaScript` files that are used to implement `AJAX` functionality in ASP.NET. These script files are sent by Web server to client browser. However, scripts forwarded by the server can be varying in accordance of functionality implemented in the Web application. Partial-page rendering also takes place with the script assistance. The Microsoft AJAX Library, in form of script, provides namespaces, inheritance, interfaces, and other related features to create your website. Sometimes while implementing `AJAX` functionality in the website, an existing script files might be insufficient to accomplish a specific task. In such cases, developers can create their own custom scripts. Such custom scripts can be saved as static `JavaScript` file or can be included as resource in an assembly. You are required to register the custom scripts with Microsoft AJAX library to use them in your application. To create culture- and language-specific applications, localization is also supported for creating applications for specific culture or language. This localization support enables you to create your own script files for various languages and culture, and save them at a remote location. You can then selectively access these script files for specific language and culture.

Web Services

A website that has AJAX functionality might need to call a Web service for implementing certain functionality. For this, you are required to add the reference of certain script files from Microsoft Library. Using the Visual studio IDE and AJAX server controls, the required script reference is automatically added to the Web page. In case you are not using the Visual Studio IDE and AJAX server controls for creating AJAX-enabled websites, you have to manually add the reference of the ASP.NET AJAX library and the required Web service. After you have added the reference of script files, the required proxy classes are automatically created, and call the Web service from the client-side. These Web services are also capable for serializing objects so that it can travel over the network. This serialization happens in the form of XML or JSON because XML and JSON are standards for cross-platform and cross-browser data transfer.

Application Services

Task of authenticating users, authorizing them specific roles, and preparing their profiles is performed through application services. These services are part of ASP.NET and can be used by any client script of an AJAX-enabled Web page. The various Web services that ASP.NET AJAX uses are as follows:

- `Profile` – Retains user information on the server
- `Membership` – Helps in authentication
- `Roles` – Implements role-based authorization for ASP.NET AJAX applications
- `Personalization` – Persists personalization data on the server
- `Globalization` – Incorporates culture-specific features in ASP.NET AJAX applications

Server Controls

ASP.NET AJAX contains a set of built-in AJAX-enabled server controls, which contains code for implementing highly interactive client behavior. ASP.NET AJAX-specific server controls provide a more server-centric approach to the application development. Adding a control ensures that supporting client script reference is added to the Web page automatically, which will be sent by the server to the client when the user accesses the Web page. You can even customize the scripts before sending them to the client, if needed. We will describe the server controls more deeply later in this chapter.

AJAX Server or Extension Controls

AJAX, first introduced as an extension to the previous versions of ASP.NET, now is a part of ASP.NET 3.5 and Visual Studio 2008. Microsoft has introduced AJAX specific controls to ease the implementation of AJAX in a website. You can find these controls under the tab named **AJAX** Extensions. When you add any of these controls to the Web form, the reference of corresponding script library is added automatically to the Web page. The list of AJAX server controls is as follows:

- The ScriptManager control
- The ScriptManagerProxy control
- The Timer control
- The UpdatePanel control
- The UpdateProgress control

Now, let's discuss these controls in detail one by one.

The ScriptManager Control

The `ScriptManager` control is the first AJAX server control in the Toolbox. This control helps in implementing the AJAX functionality in the ASP.NET website. You need to place this control on the webpage wherever AJAX functionality is required. In the absence of this control, no other AJAX server controls, such as `Timer`, `UpdatePanel`, and `UpdateProgress`, can be added to the Web page. The `ScriptManager` control is responsible for managing client scripts for AJAX-enabled website. The `ScriptManager` control uses a `ScriptManager` class to manage AJAX script libraries and script files. The inheritance hierarchy of the `ScriptManager` class is as follows:

```
System.Object
    System.Web.UI.Control
        System.Web.UI.ScriptManager
```

The `ScriptManager` control renders the AJAX Library scripts to the browser and supports partial page rendering, Web-service calls, and use of ASP.NET AJAX Client Library. You can add the `ScriptManager` control in the Web page by just dragging the control from the Toolbox and dropping it on the design page. The `ScriptManager` control is initialized using the following mark up code snippet:

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
```

The noteworthy properties of the `ScriptManager` class are listed in Table 32.3:

Table 32.3: Noteworthy Properties of ScriptManager Class

Property	Description
`AllowCustomErrorsRedirect`	Specifies a value to indicate whether or not to use the custom error section of the web.config file when an error occurs during the asynchronous `postback`
`AsyncPostBackErrorMessage`	Specifies an error message that is returned to the client when an unhandled server exception occurs during an asynchronous `postback`
`AsyncPostBackSourceElemenetID`	Specifies the ID of the control that triggers the asynchronous `postback`
`AsyncPostBackTimeout`	Specifies a time value in seconds to timeout the asynchronous `postback` if the response is not received
`AuthenticationService`	Retrieves the `AuthenticationServiceManager` object that is associated with the current `ScriptManager` instance
`EnablePageMethods`	Specifies a value to indicate whether or not to call the page method in an ASP.NET page from the client script
`EnablePartialRendering`	Specifies a value to indicate whether or not to enable partial-page rendering
`EnableScriptGlobalization`	Specifies a value to indicate whether or not the `ScriptManager` control renders script that supports parsing and formatting of different culture-specific information
`EnableScriptLocalization`	Specifies a value to indicate whether or not the `ScriptManager` control renders the localized versions of scripts
`IsDebuggingEnabled`	Specifies whether or not the `ScriptManager` control renders the debug versions of client script libraries
`IsInAsyncPostBack`	Specifies whether or not the `ScriptManager` control executes the current `postback` in a partial-rendering mode
`LoadScriptsBeforeUI`	Specifies a value to indicate whether scripts are loaded before or after the page markup
`ProfileService`	Retrieves the `ProfileServiceManager` object that is associated with the current `ScriptManager` instance
`RoleService`	Retrieves the `RoleServiceManager` object that is associated with the current `ScriptManager` instance
`ScriptMode`	Specifies a value to indicate whether the `ScriptManager` control renders the debug or release versions of client script libraries
`ScriptPath`	Specifies the location of the root path that is used to build paths to ASP.NET AJAX and custom script files
`Scripts`	Retrieves a `ScriptReferenceCollection` object having multiple `ScriptReference` objects, each represents a script file rendered to the client
`Services`	Retrieves a `ServiceReferenceCollection` object that contains a `ServiceManager` object for each Web service that ASP.NET exposes on the client for `AJAX` functionality

Table 32.3: Noteworthy Properties of ScriptManager Class	
SupportsPartialRendering	Specifies a value to indicate whether or not the client supports partial-page rendering
Visible	Overrides the Visible property of the Control class to prevent the setting of the value of Visible property of ScriptManager class

The noteworthy methods of the ScriptManager class are listed in Table 32.4:

Table 32.4: Noteworthy Methods of ScriptManager Class	
GetCurrent	Retrieves the ScriptManager instance for a given Web page object.
GetRegisteredArrayDeclarations	Retrieves a read-only collection of ECMAScript (JavaScript) array declarations that were previously registered with the Web page object.
GetRegisteredClientScriptBlocks	Retrieves a read-only collection of client script blocks that were previously registered with the ScriptManager control.
GetRegisteredDisposeScripts	Retrieves a read-only collection of dispose scripts that were previously registered with the Web page object.
GetRegisteredExpandoAttributes	Retrieves a read-only collection of custom attributes that were previously registered with the Web page object.
GetRegisteredHiddenFields	Retrieves a read-only collection of hidden fields that were previously registered with the Web page object.
GetRegisteredOnSubmitStatements	Retrieves a read-only collection of onsubmit statement that was previously registered with the Web page object.
GetRegisteredStartupScripts	Retrieves a read-only collection of startup scripts that were previously registered with the Web page object.
RegisterArrayDeclaration	Registers an ECMAScript (JavaScript) array declaration with the ScriptManager control and adds the array to the Web page. It is an overloaded method.
RegisterAsyncPostBackControl	Registers a control as a trigger for asynchronous postbacks.
RegisterClientScriptBlock	Registers a client script block with the ScriptManager control and adds the script block to the Web page. It is an overloaded method.
RegisterClientScriptInclude	Registers a client script file with the ScriptManager control and adds the reference of the script file to the Web page. It is an overloaded method.
RegisterClientScriptResource	Registers client script that is embedded in an assembly with the ScriptManager control for use with a control participating in partial-page rendering. It is an overloaded method.
RegisterDataItem	Sends custom data to control during partial-page rendering. It is an overloaded method.
RegisterDispose	Register a dispose script for a control that is inside an UpdatePanel control.
RegisterExpandoAttribute	Registers a name/value pair with the ScriptManager control as a custom attribute of the specified control.
RegisterExtenderControl	Registers an extender control with the current ScriptManager instance.
RegisterHiddenField	Registers a hidden field. It is an overloaded method.
RegisterOnSubmitStatement	Registers ECMAScript (JavaScript) code that is executed on submitting the Web form. It is an overloaded method.
RegisterPostBackControl	Registers a control as a trigger for a postback.
RegisterScriptControl	Registers a script control with the current ScriptManager instance.

Table 32.4: Noteworthy Methods of ScriptManager Class	
RegisterScriptDescriptors	Instructs the ScriptManager control to call back to the ScriptControl or ExtenderControl classes to return scripts that support the client object. It is an overloaded method.
RegisterStartupScript	Registers a startup script block to the ScriptManager control and adds the script block to the Web page. It is an overloaded method.
SetFocus	Specifies the browser focus to the specified control. It is an overloaded method.

Noteworthy events of the ScriptManager class are listed in Table 32.5:

Table 32.5: Noteworthy Events of ScriptManager Class	
AsyncPostBackError	Occurs when a page error is found during an asynchronous postback
ResolveScriptReference	Occurs when a member of the Scripts() collection is registered with the ScriptManager control

Differentiating Between AJAX and Non-AJAX Applications

ASP.NET 3.5 provides us with AJAX-specific controls to create AJAX-enabled website. Along with these controls, we can use other ASP.NET controls, such as Standard controls, Login controls, Navigation controls, and the rest. To understand AJAX better, you need to understand the difference in the functionalities of AJAX and non-AJAX applications. For this purpose, two examples are given. We first create a simple Web application, and then extend it to implement the AJAX functionality in the application.

Creating a Simple non-AJAX Application

Now, you are going to create a simple ASP.NET website that displays current system time on a button click. In this application, we have added two labels and a button on the Web page. The first label is displayed as the heading and second label displays the time, when the button is clicked. You can find this code of Simple_ApplicationVB application in the Code\ASP.NET\Chapter 32\Simple_ApplicationVB folder on the CD. Listing 32.1 shows the complete code for the Default.aspx page of simple ASP.NET website:

Listing 32.1: Showing the Code for the Default.aspx Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
    Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Simple Non AJAX Application</title>
    <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <div id="header">

        </div>
        <div id="sidebar">
            <div id="nav">
                 
            </div>
        </div>
    </div>
    <div>
    <br />
```

```
        <h2>
            <asp:Label ID="LabelHadding" runat="server" Text="Simple Application
            Without AJAX"></asp:Label>
        </h2>
        <asp:Label ID="TimeLabel" runat="server" Text="Click the Button to Display
        Current System Time" ></asp:Label>
         
        <asp:Button ID="Button1" runat="server" Text="Button" />
        <div id="footer">
            <p class="left">
                All content copyright &copy; Kogent Solutions Inc.
            </p>
        </div>
    </div>
    </form>
</body>
</html>
```

In Listing 32.1, you can see that there is a label and a button on the form. When you click the Button control, system's current date and time is displayed in the place of the label. The button_click event is handled in the code-behind file. Listing 32.2 shows the complete code for the code-behind file of `Default.aspx` page of simple ASP.NET website:

Listing 32.2: Showing the Code for the Code-Behind File

```
Partial Class _Default
    Inherits System.Web.UI.Page
    Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Handles Button1.Click
        Dim myDateTime As String
        myDateTime = System.DateTime.Today.ToLongDateString()+ " "
        myDateTime += System.DateTime.Now.ToLongTimeString()
        TimeLabel.Text = myDateTime
    End Sub
End Class
```

Now, let's run the application by pressing the F5 key. The output is as shown in Figure 32.6:

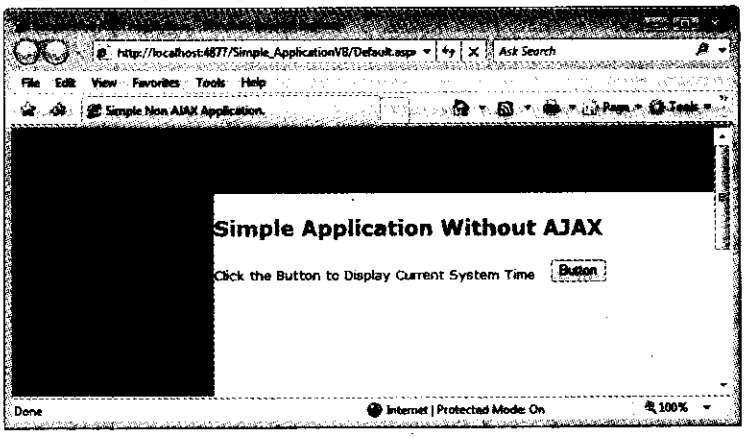


Figure 32.6: Application without AJAX at First Run

Now, click the Button control. The `Click the Button to Display Current System Time` label is converted into System's current Date and Time, as shown in Figure 32.7. Every time you click Button, the entire page gets refreshed to display the current Date and Time of the system:

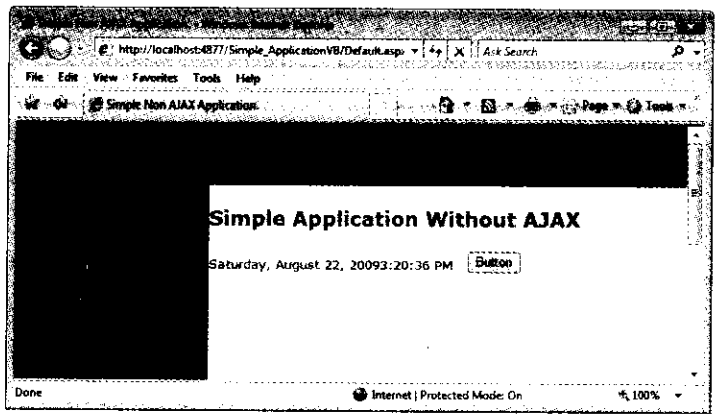


Figure 32.7: Output After Button Click

Creating a simple AJAX Application

You have created a non-AJAX application that displays the current date and time of the server. Now, you will extend the same application to create a new AJAX-enabled application. This application also displays the system's current date and time, but there is no page flash. In short, the process of getting the current date and time will take place in the background because of AJAX. In this application, you will learn the use of `the ScriptManager` and `UpdatePanel` controls. The code lines using `the ScriptManager` and `UpdatePanel` controls are highlighted. You can find this code of `Simple_ApplicationWithAJAXVB` application in the Code\ASP.NET\Chapter 32\`Simple_ApplicationWithAJAXVB` folder on the CD. Listing 32.3 shows the complete code for the `Default.aspx` page of simple AJAX application:

Listing 32.3: Showing the Code for the `Default.aspx` Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>Simple AJAX Application </title>
  <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form id="form1" runat="server">
  <div>
      <div id="header">
      </div>
      <div id="sidebar">
          <div id="nav">
               
          </div>
      </div>
  </div>
    <div>
      <br />
```

```
        <div id="footer">
            <p class="left">
                All content copyright &copy; Kogent Solutions Inc.
            </p>
        </div>
    </div>
    </form>
</body>
</html>
```

Now, change the code-behind file code. Listing 32.4 shows the complete code for the code-behind file of `Default.aspx` page of simple AJAX application:

Listing 32.4: Showing the Code for the Code-Behind File

```
Partial Class _Default
    Inherits System.Web.UI.Page
    Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Handles Button1.Click
        Dim myDateTime As String
        myDateTime = System.DateTime.Today.ToLongDateString() + " "
        myDateTime += System.DateTime.Now.ToLongTimeString()
        TimeLabel.Text = myDateTime
    End Sub
End Class
```

Now, let's run the application by pressing the F5 key. The output is as shown in Figure 32.8:

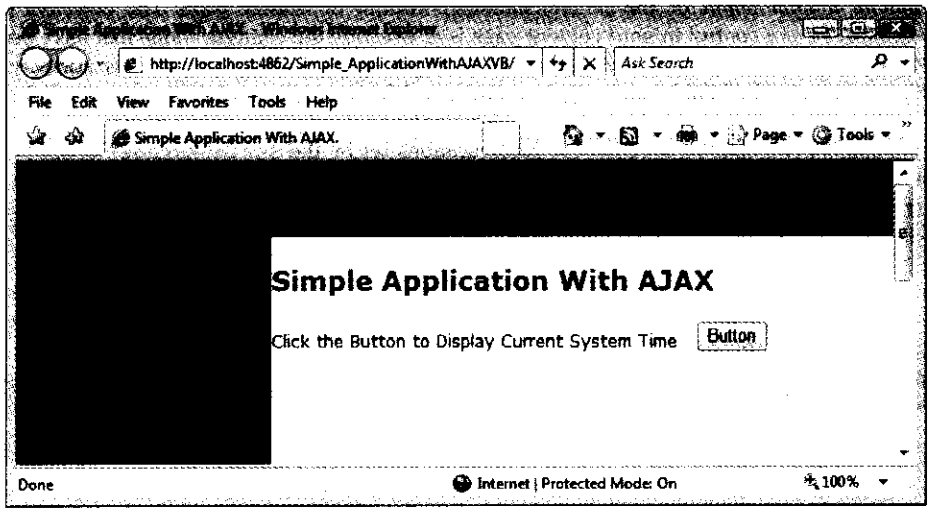


Figure 32.8: Output of Simple_Application with AJAX at First Run

As shown in Figure 32.8, click the Button control. It will convert the label `Click the Button to Display Current System Time` into the system's current date and time as shown in Figure 32.9:

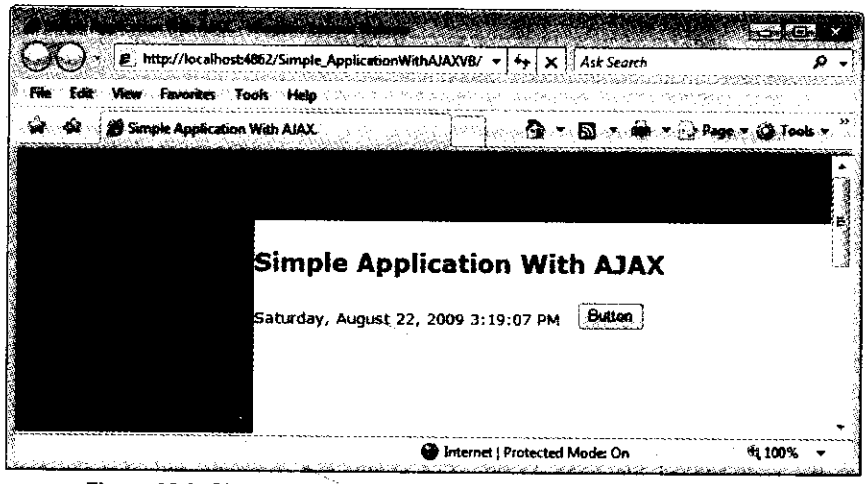


Figure 32.9: Simple_Application with AJAX Displaying Date and Time

Notice that there is no page flash because of AJAX functionality. You will also not find any action in the progress bar as happened in the earlier example. All of this is possible with the help of AJAX Server controls added in the Default.aspx file.

The ScriptManagerProxy Control

As you know, an AJAX-enabled website can contain only one ScriptManager control (on each Web page), which manages all the AJAX Server controls on that page. Additionally, if the website contains a Master page and the ScriptManager control is placed in the Master page, then all the content pages (Web page inherited from the Master page) use the ScriptManager control of the Master Page to manage its AJAX server controls. For this, each content page requires a link to connect to Master page's ScriptManager control. The content page uses the ScriptManagerProxy control to make a link. The ScriptManagerProxy control of the content page works as a local object of the ScriptManager control of Master page. The objective of using the ScriptManagerProxy control in your content page is to override the configurations that have been set by the ScriptManager control of the Master page. For example, if the services and scripts added earlier in the ScriptManager control of the Master page are not useful for your current Web page, you can remove them through the ScriptManagerProxy control. The ScriptManagerProxy control uses a ScriptManagerProxy class to override the configurations set by the ScriptManager control of the Master page. The inheritance hierarchy of the ScriptManagerProxy class is shown in the following code:

```
System.Object
    System.Web.UI.Control
        System.Web.UI.ScriptManagerProxy
```

You can add the ScriptManagerProxy control in the Web page by just dragging the control from the Toolbox and dropping it on the design page. The ScriptManagerProxy control is initialized using the following mark up code snippet:

```
<asp:ScriptManagerProxy ID="ScriptManagerProxy1" runat="server">
</asp:ScriptManagerProxy>
```

The properties, methods and events of the ScriptManagerProxy class are same as of the ScriptManager class.

The Timer Control

Sometimes, there can be a situation when you want to update the contents of the page after a specific interval. For example, there is a Web page showing the stock prices where the price of stocks changes frequently. You are asked to update the stock prices after every minute. In this scenario, you need to use the Timer control. The Timer control contains a property named Interval and an event named Tick. The Interval property of

the Timer control is used to specify the desired time limit in Seconds; in our example, it is 1 minute, that is, 60 seconds. When this time limit is over, it raises the Tick event. You can handle this Tick event to update the content of your Web page; in this example, it is stock price. A single AJAX-enabled Web page can contain more than one part; it is your choice whether to use single Timer control for all the parts of Web page or to use different Timer controls for every part of a Web page. Moreover, a single Timer control can refresh the contents of more than one parts of a Web page. The Timer control's another useful property is the Enabled property. Using the Enabled property of the Timer control, you can turn the Timer control ON or OFF at runtime or design time. The Enabled property controls the Tick event from firing. If the Enabled property is True, then the Timer control can fire the Tick event, otherwise not. The Timer control is the instance of the Timer class. The inheritance hierarchy of the Timer class is as follows:

```
System.Object
  System.Web.UI.Control
    System.Web.UI.Timer
```

You can add the Timer control in the Web page by just dragging the control from the Toolbox and dropping it on the design page. The Timer control is initialized using the following mark up code snippet:

```
<asp:Timer ID="Timer1" runat="server" Interval="50000">
    </asp:Timer>
```

Noteworthy properties of the Timer class are listed in Table 32.6:

Table 32.6: Noteworthy Properties of Timer Class

Property	Description
Enabled	Specifies a value to indicate whether or not the Timer control initiates a postback when the interval time has elapsed
Interval	Specifies a time value in milliseconds to wait before the Timer control initiates a postback
Visible	Overrides the Visible property of the Control class to prevent this value from being set

The UpdatePanel Control

Partial-page rendering is the feature of AJAX-enabled websites, which enables you to update a specific part, instead of the entire Web page. The UpdatePanel control helps to divide your Web page into parts; where each part can be updated independently with each other. The UpdatePanel control uses a synchronous postback for refreshing the selected part of the page; but inbuilt client script transforms this synchronous request to asynchronous request with the help of the XMLHttpRequest object, so that it can update the information without the page flash. A single Web page can contain multiple UpdatePanel controls where each control can create a separate part of your Web page. You need not to write any custom client script for partial page rendering when you are using the UpdatePanel control. The UpdatePanel control uses the UpdatePanel class to support partial-page rendering. The inheritance hierarchy of the UpdatePanel class is displayed in the following code:

```
System.Object
  System.Web.UI.Control
    System.Web.UI.UpdatePanel
```

You can add the UpdatePanel control in the Web page by just dragging the control from the Toolbox and dropping it on the design page. The UpdatePanel control is initialized using the following mark up code snippet:

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
  <ContentTemplate>
      <!-- PAGE REGIONS TO BE REFRESHED -->
      <!-- PAGE CONTENTS -->
  </ContentTemplate>
</asp:UpdatePanel>
```

Noteworthy properties of the UpdatePanel class are listed in Table 32.7:

Table 32.7: Noteworthy Properties of UpdatePanel Class	
ChildrenAsTriggers	Specifies a value to indicate whether or not triggering the postbacks from immediate child controls of an UpdatePanel control updates the panel's content
ContentTemplate	Specifies a template to define the content of the UpdatePanel control
ContentTemplateContainer	Retrieves a control object to which you can programmatically add child controls
Controls	Retrieves the ControlCollection object having the child controls for the UpdatePanel control
IsInPartialRendering	Specifies a value to indicate whether or not to update the UpdatePanel when a client triggers the asynchronous postback
RenderMode	Specifies a value to indicate whether or not the content of the UpdatePanel is placed within the HTML or tags
Triggers	Retrieves an UpdatePanelTriggerCollection object having AsyncPostBackTrigger and PostBackTrigger objects that have been defined for the UpdatePanel control
UpdateMode	Specifies a value to indicate when the content of the UpdatePanel control is updated

An asynchronous postback is quite similar to the regular postback where the resulting server page executes the entire page. However, the asynchronous postback executes the limited page regions that are enclosed within the UpdatePanel control.

Using AJAX Server Controls

Now, you are familiar with a simple AJAX application and have understood the basic idea behind AJAX. You have also used the ScriptManager and UpdatePanel controls in the previous application of this chapter. Let's discuss the other AJAX Server controls, such as:

- Timer control
- UpdateProgress control

Now, let's start our discussion with the Timer control.

Timer Control

The Timer control is used for updating contents of the UpdatePanel control at predefined intervals. Single Timer control can refresh more than one UpdatePanel control simultaneously. The Timer control has an Interval property for defining the time limit after which the content gets refreshed. When this interval limit gets over, the Tick event is raised. You can handle this event for updating the content values defined in the UpdatePanel control. The Timer control can be used either inside the UpdatePanel control or outside the UpdatePanel control. Now, let's try to understand the UpdatePanel control in both ways.

Timer Control Inside the UpdatePanel Control

To use the Timer control inside the UpdatePanel Control, drag-and-drop from the Toolbox as a sub-element of the <ContentTemplate> tag and follow the following steps:

1. Set the interval property of the Timer control as 1000 that is equal to 1 second as it takes value in milliseconds.
2. Now, double-click the Timer control in design mode. It will generate event handler function in the code-behind file to handle the Tick event.

You can find this code of `TimerControlAppVB` application in the Code\ASP.NET\Chapter 32\ `TimerControlAppVB` folder on the CD. Listing 32.5 shows the complete code for the `Default.aspx` page::

Listing 32.5: Showing the Code for the `Default.aspx` Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
    Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Timer Control Example</title>
    <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <div id="header">

        </div>
        <div id="sidebar">
            <div id="nav">
                 
            </div>
        </div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <asp:UpdatePanel ID="UpdatePanel1" runat="server">
        <ContentTemplate>
        <asp:Timer ID="TimeUpdater" runat="server" Interval="1000"
         ontick="TimeUpdater_Tick">
        </asp:Timer>
        <h2>
        <asp:Label ID="LabelHadding" runat="server" Text="Application Using Timer
        Control"></asp:Label>
        </h2>
        <asp:Label ID="TimeLabel" runat="server" Text="Current System Time"></asp:Label>
         
        </ContentTemplate>
        </asp:UpdatePanel>
        <div id="footer">
        <p class="left">
            All content copyright &copy; Kogent Solutions Inc.
        </p>
        </div>
    </div>
    </form>

</body>
</html>
```

You need to update the time every second, similar to the clock. For this, you have to handle the `Tick` event of the `Timer` control. For this purpose, you need to write the code in the code-behind file. Listing 32.6 shows the complete code for the code-behind file of `Default.aspx` page::

Listing 32.6: Showing the code for the Code-Behind File

```
Partial Class _Default
    Inherits System.Web.UI.Page
```

```
    Protected Sub TimeUpdater_Tick(ByVal sender As Object, ByVal e As System.EventArgs)
     Handles TimeUpdater.Tick
    TimeLabel.Text = System.DateTime.Now.ToLongTimeString
    End Sub
End Class
```

Now, let's run the application by pressing the F5 key. The output is as shown in Figure 32.10:

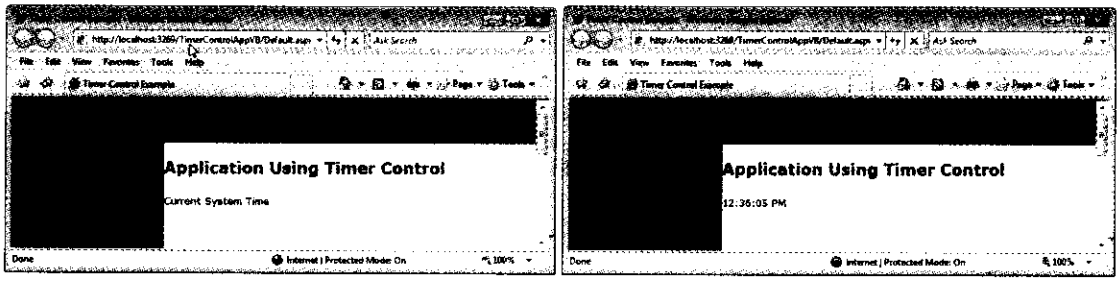


Figure 32.10: Displaying Output Before Firing Tick Event (Left Side) and After Firing Tick Event (Right Side)

The preceding figure, on the left side, displays the output before occurring `Tick` Event for the first time and, on the right side, it displays the output after occurring the `Tick` Event that converts the label into the current time of the system.

Timer Control **Outside** the **UpdatePanel** Control

You can use the `Timer` control outside the `UpdatePanel` control. This approach is appropriate when there are more than one `UpdatePanel` control in the same Web page and you need to use same `Timer` control for all or some of them. To do so, you need to perform some changes in your application. However, these changes will not affect the output of your application. First, drag-and-drop the `Timer` control from the toolbox. Drop it after the `ScriptManager` control and before the `UpdatePanel` control; and configure its properties in the following way:

- The ScriptManager Tag:

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
```

- The `Timer` control outside the `UpdatePanel` control and after the `ScriptManager Tag:`

```
<asp:Timer ID="Timer1" runat="server" Interval="1000"
        ontick="TimeUpdater_Tick">
</asp:Timer>
```

Configure the `UpdatePanel` Control's `<Triggers>` Element for referencing the `Timer` control in the following way:

```
<asp:UpdatePanel ID="updatePanel1" runat="server">
    <Triggers>
        <asp:AsyncPostBackTrigger ControlID="Timer1" EventName="Tick" />
    </Triggers>
```

The UpdateProgress Control

While working with the `UpdatePanel` control, sometimes you need to show the status of the information requested by the user. In other words, we can say that we need to make our viewer aware of the progress status of the requested data to refresh the content of the `UpdatePanel` control. This task is accomplished with the help of the `UpdateProgress` control. As per your requirement, you can use image, text, or a progress bar to show the status. This control is basically used in situations when your page contains more than one `UpdatePanel` control and task of updating content is performed at different times. This control is also useful when speed of updating content is slow. In this condition, user gets idea about how much information is processed and for how much time, the user needs to wait. The `UpdateProgress` control is visible, while the task of updating content is in progress, as soon as the content gets updated, the `UpdateProgress` control disappears. The

UpdateProgress control uses the UpdateProgress class to support the properties involved in displaying the update progress. The inheritance hierarchy of the UpdateProgress class is as follows:

```
System.Object
  System.Web.UI.Control
    System.Web.UI.UpdateProgress
```

You can add the UpdateProgress control in the Web page by just dragging the control from the Toolbox and dropping it on the design page. The UpdateProgress control is initialized using the following mark up code snippet:

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server"
 AssociatedUpdatePanelID="UpdatePanel1">
  <ProgressTemplate>
      Please wait for a while. Page is Updating.....
  </ProgressTemplate>
</asp:UpdateProgress>
```

Noteworthy properties of the UpdateProgress class are listed in Table 32.8:

Table 32.8: Important Properties of UpdateProgress Class

[illegible]	[illegible]
AssociatedUpdatePanelID	Specifies the ID of the UpdatePanel control for which the status has been displayed
DisplayAfter	Specifies a time value in milliseconds before which the UpdateProgress control is displayed
DynamicLayout	Specifies a value to indicate whether or not to render the progress template dynamically
ProgressTemplate	Specifies a template that defines the content of the UpdateProgress control

NOTE

For working with the UpdatePanel, UpdateProgress, and Timer Controls, you need to have the ScriptManager control on the Web page, because the ScriptManager control contains a reference to the built-in script libraries for implementing AJAX functionality.

UpdateProgress Control

The greatest advantage of AJAX is to update Web page content asynchronously. Sometimes, it creates the problem in the scenarios, when the updating speed is very slow or data, which needs to be updated, is quite heavy. In such a case, the user does not get any idea whether the request is being processed or not. To resolve this problem, we need to use the UpdateProgress control to show the status of request that is processed. You can use single UpdateProgress control either for single UpdatePanel control or for multiple UpdatePanel controls. Now, let's try to understand it in both ways.

Using UpdateProgress Control for Single UpdatePanel Control

Here is an example using the UpdateProgress control for single UpdatePanel control. In this example, the UpdateProgress control has been added in the UpdatePanel control. We have set UpdateProgress control's AssociatedUpdatePanelID property as UpdatePanel1. Here, UpdatePanel1 is the ID of the UpdatePanel control for which the UpdateProgress control works. The UpdateProgress control contains a sub-element named ProgressTemplate, which contains text to display when updation is in progress. You can find this code of UpdateProgAppVB application in the Code\ASP.NET\Chapter 32\UpdateProgAppVB folder on the CD. Listing 32.7 shows the complete code for the Default.aspx page of UpdateProgress control:

Listing 32.7: Showing the Code for the Default.aspx Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
 Inherits="_Default" %>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>UpdateProgress Control Example</title>
  <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form id="form1" runat="server">
  <div>
      <div id="header">

      </div>
      <div id="sidebar">
          <div id="nav">
               
          </div>
      </div>
      <asp:ScriptManager ID="ScriptManager1" runat="server">
      </asp:ScriptManager>
      [redacted]
      <h2>
      <asp:Label ID="LabelHadding" runat="server" Text="Application Using
      UpdateProgress Control">
      </asp:Label>
      </h2>
      <asp:Label ID="Label1" runat="server" Text="UpdatePrgress Demo">
      </asp:Label>
           
      <asp:Button ID="Button1" runat="server" Text="Button" />
      </ContentTemplate>
      </asp:UpdatePanel>
      <div id="footer">
          <p class="left">
              All content copyright &copy; Kogent Solutions Inc.
          </p>
      </div>
  </div>
  </form>
</body>
</html>
```

Now, change the code-behind file code. Listing 32.8 shows the complete code for the code-behind file of `Default.aspx` page of `UpdateProgress` control:

Listing 32.8: Showing the Code for the Code-Behind File

```
Partial Class _Default
  Inherits System.Web.UI.Page
      Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
  Handles Button1.Click
  System.Threading.Thread.Sleep(5000)
  Label1.Text = "Demonstration on UpdateProgress Control taken place last time at " +
  System.DateTime.Now.ToLongTimeString
```

```
    End Sub
End Class
```

Now, let's run the application by pressing the F5 key. Then, click the `Button` button. The output is as shown in Figure 32.11:

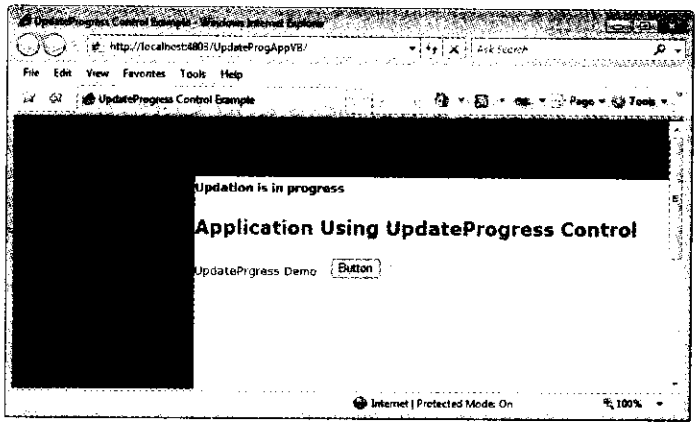


Figure 32.11: Showing UpdateProgress Control

When you click the Button control, you will see a label showing `Updation is in progress` at the top of the page (Figure 32.11). As soon as the label `UpdateProgress Demo` changes its value, this label disappears as shown in Figure 32.12:

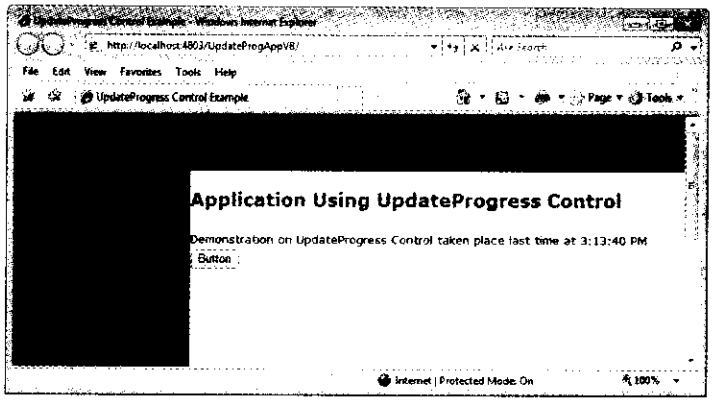


Figure 32.12: Showing Final Output of UpdateProgress Control

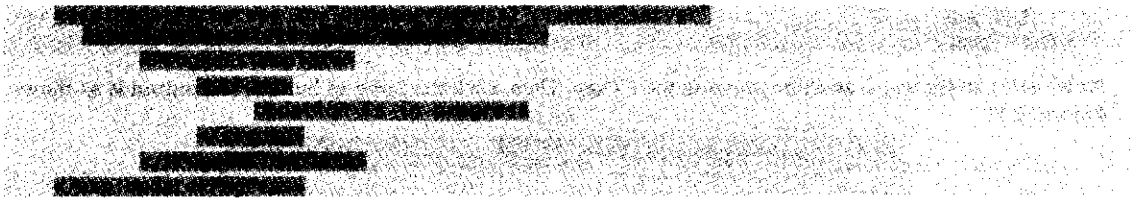
NOTE

In Listing 32.8 code, we have intentionally caused a delay through the code line System.Threading.Thread.Sleep(5000) to make you understand the use of the UpdateProgress Control, However, in real-time application, this will be caused due to some genuine reasons, such as Network traffic, and the rest.

Using UpdateProgress Control for Multiple UpdatePanel Control

To use single `UpdateProgress` control for multiple `UpdatePanel` controls, do not place the `UpdateProgress` control inside a particular `UpdatePanel` control as you did in the previous application. Instead, place the `UpdateProgress` control just after the `<ScriptManager>` control. The rest of the code will remain the same as in the previous application. See the following example, where the changes are needed:

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
```



ASP.NET AJAX Control Toolkit Extenders and Controls

The ASP.NET AJAX Control Toolkit (ACT) consists of new controls that have AJAX functionalities and a variety of extenders. These extenders are attached to other ASP.NET controls to enhance the control's functionality. Using the ACT, you can remove the limitation of ASP.NET controls and include new functionalities to it. The extender controls are derived from the `System.Web.UI.ExtenderControl` abstract base class and enables the developers to encapsulate UI behavior, and make it easy to add richer functionality to an application. For example, if you add a standard ASP.NET `Button` control on the Web Forms Designer, an `Add Extender` link appears in the Button's smart tag, as shown in Figure 32.13:

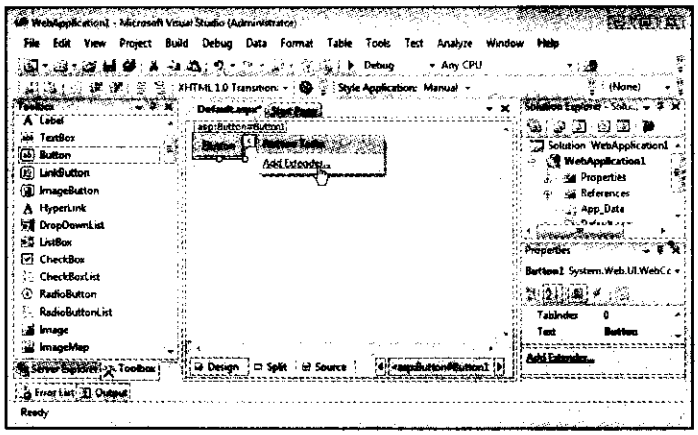


Figure 32.13: Adding Extender to Button Control

When the Add Extender link is clicked, the Extender Wizard appears displaying a list of extenders that can be applied to the `Button` control, as shown in Figure 32.14:

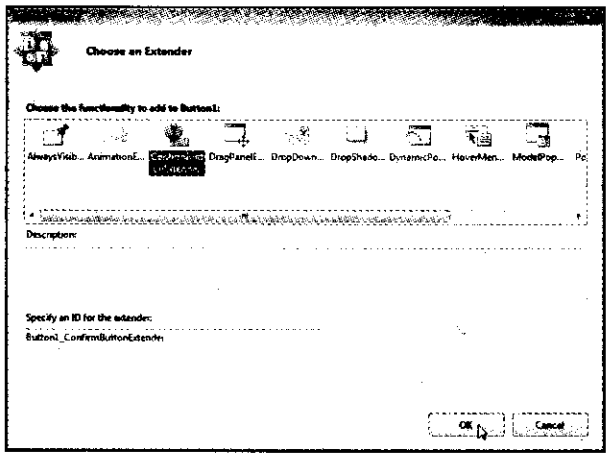


Figure 32.14: Selecting an Extender from a List

NOTE

The extenders may vary for the different controls; i.e., when you use a TextBox control on the Web Form and click Add Extender for the TextBox control, the ConfirmButtonExtender does not appear, because this extender can only be applied to a Button control.

Along with a vast set of extenders, ASP.NET ACT also includes a few traditional server controls with rich capabilities, such as `Accordion`, `TabContainer`, and `Rating`. A control offers a closed set of functionalities that have a well-known and fixed behavior that you can implement through properties and events. An extender mostly offers a client-side behavior that you can attach with the existing ASP.NET controls to experience the enhanced functionalities. With this logical difference, both server controls and extenders are implemented in the same way and finally, both represent a server control.

The ASP.NET ACT consists of some new controls that have the AJAX functionality and a variety of control extenders that are attached to another ASP.NET controls to enhance the control's functionality. Table 32.9 lists the controls of the ASP.NET ACT and their brief description:

Table 32.9: ASP.NET AJAX Control Toolkit

Name	Description
`Accordion and AccordionPane`	Enable you to provide multiple panes simultaneously in the Web page among which only one pane can be displayed at a time. It is very much similar to a set of CollapsiblePanelExtender control appears within a Web page where expanding one AccordionPane control results in collapsing the other remaining panes in the Web page.
`AlwaysVisibleControlExtender`	Allows you to pin or fix an ASP.NET control, or a set of controls, to one of the page corners. This makes a floating appearance of the fixed controls over the background body content, when you scroll or resize the window.
`AnimationExtender`	Enables you to use built-in animation framework to create very sophisticated effects in an easy and declarative fashion. This allows you to add a broad set of fancy animation effects to your website.
`AutoCompleteExtender`	Enables you to add a set of suggestions that is displayed when a user starts typing in the textbox. For example, as you start typing a word into the textbox, a list of suggested word appears in the associated control, basically a popup panel, with the prefix typed into the textbox.
`CalendarExtender`	Enables you to display a calendar within a textbox. With this, you can select a date, month, and year with a customizable date format in a popup control.
`CascadingDropDown`	Enables you to make the list of items in one DropDownList control depending on the selected items in other DropDownList control. When you select an item in the parent DropDownList control, the items in the child DropDownList control is automatically populated by performing an asynchronous postback.
`CollapsiblePanelExtender`	Enables you to hide or display the contents of a panel alternatively. This adds collapsible sections to your Web page. When you click the header of the collapsible section, the contents either appear or disappear.
`ConfirmButtonExtender`	Enables you to attach a confirmation box to a button control. When you click on the button, a message appears within the confirmation dialog box. The confirmation dialog box is the default JavaScript confirmation box but you can also associate a modal dialog box with the button control.
`DragPanelExtender`	Enables you to add a drag effect to the panels on you Web page. This makes the panels virtually floating on the Web page. Using this extender control, you can easily drag the panels by clicking their associated drag handles.
`DropDownExtender`	Enables you to create a SharePoint-style drop-down menu by attaching to any of the ASP.NET controls. When you click the control to which this extender is attached, the menu appears in another panel or control.

Table 32.9: ASP.NET AJAX Control Toolkit

Name of the Control	Description
DropShadowExtender	Enables you to add a drop shadow to an ASP.NET panel control. Using this extender, you can specify the width and opacity of the drop shadow as well as make the corners of the panel round.
DynamicPopulateExtender	Enables you to send an asynchronous call to the server that dynamically populates the contents of a control; that means it replaces the contents of a control with the result of a Web service or page method call. You can also set another control, such as button control, to trigger the asynchronous request.
FilteredTextBoxExtender	Enables you to apply filtering to a TextBox control that result to prevent certain characters from being inserted in the text box. For example, you can limit a text box to either enter only characters, or digits, or dates.
HoverMenuExtender	Enables you to display a popup menu when you hover over another control. This extender attaches to an ASP.NET control and associates that particular control with a popup panel that displays additional content. Using the HoverMenuExtender control, you can also specify the position of the popup panel in the Web page.
ListSearchExtender	Enables you to perform an incremental search for the desired items in the ListBox or DropDownList control by typing the word incrementally.
MaskedEditExtender	Enables you to restrict a user to enter only a certain pattern of characters in the text box by applying a mask to the input.
MaskedEditValidator	Enables to verify whether or not the input text matches with the pattern specified in the MaskedEditExtender control when the MaskedEditValidator control is attached to the MaskedEditExtender control and its associated TextBox control.
ModalPopupExtender	Enables you to display page content to the user in a modal manner by displaying a modal popup. When the modal popup appears, the remainder of the Web page grays out, preventing you from interacting with the page. However at that time, you can interact with the Modal popup contents.
MutuallyExclusiveCheckBoxExtender	Enables you to treat a set of CheckBox controls similar to a set of RadioButton controls. You can use this extender when multiple choices are available from which you can choose only one choice. However, these check boxes are different from the radio buttons as they can be cleared which is not possible with radio button.
NoBot	Enables you to apply anti-bot techniques, such as enforcing a configurable delay between when a form is requested and when it can be posted back, to input forms that prevents spam bots from posting advertisements to your website. This extender is mostly applicable in stopping spam.
NumericUpDownExtender	Enables you to add up and down buttons next to the TextBox control. These buttons are used to cycle (increment or decrement) through a set of numeric values or other items, such as months of the year. However, you can also use custom images in place of the up and down buttons to navigate through the numeric values.
PagingBulletedListExtender	Enables you to display diffenent contents depending on the bullet clicked. This extender is attached to a BulletedList control and provides client-side sorted paging. Using this extender you can specify the number of characters that you want to use for creating heading indices.

Table 32.9: ASP.NET AJAX Control Toolkit

	Description
PasswordStrength	Enables you to display a popup control to display the security strength of the password. This extender is attached to a TextBox control and when the user enters the password in the TextBox control. The strength is calculated depending on the specified complexity level of the chosen password.
PopupControlExtender	Enables you to display a popup window that has additional contents, such as a calendar on a TextBox control in which the user is expected to enter a date. This extender is attached to any ASP.NET control in the Web page, which you can click to open the popup window. The contents of the popup window are expressed by using the Panel control, which is probably within the UpdatePanel control.
Rating	Enables you to apply rating option to an item on the Web page. This extender allows user to select the number of stars that represent the rating of an item. Using this extender, you can also specify the initial and maximum rating that a user is allowed to rate an item.
ReorderList	Enables you to display an interactive list of items that can be reordered through drag and drop. As the item is dropped to its new location, the data source gets updated automatically.
ResizableControlExtender	Enables you to dynamically resize the ASP.NET controls, such as panels of texts or images. This extender is attached to any ASP.NET control on a Web page to allow resizing that control using a handle attached to the lower-right corner of the control. The user resizes the control same as a window and just after resizing the control, the content within the control automatically resizes to fit to the new dimension.
RoundedCornersExtender	Enables you to add rounded corners around the ASP.NET controls in the Web page. This is a subset of DropShadowExtender control, as it is limited to round the corners of child panels. Instead of making all the corners round, you can apply setting to round either one, or two, or three corners.
SliderExtender	Enables you to deform a classic TextBox control into a graphical slider. The user uses this slider to pick up a numeric value from a finite range. You can set the orientation of the slider either horizontal or vertical and specify a discrete interval of values that the slider accepts. The default range of the slider is 0 to 100.
SlideShowExtender	Enables you to create a slide show of images. This extender allows you to add button to hit previous, next, and play. You can play the slide show automatically on page load and apply loop play to the images. The images included in the slide show are added to the extender using a page method or a Web service.
TabContainer	Enables you to create a tabbed interface where clicking on each tab displays different contents. Using this extender, you can present your Web page in multiple views.
TextBoxWatermarkExtender	Enables you to display a default background text in the text box when the text box is empty, which is called a Watermark effect. When you select the text box to start typing your text, the background text disappears.
ToggleButtonExtender	Enables you to customize the appearance of the CheckBox control. Since check boxes represent graphical controls that are visually represented by a pair of little bitmaps (generally selected and unselected) plus companion text, you can use custom images in place of the check boxes.

Table 32.9: ASP.NET AJAX Control Toolkit	
ToolkitScriptManager	Enables you to add the ASP.NET ACT controls on the Web page. Before the introduction of this control, the ScriptManager control of the AJAX Extension controls was added to the Web page. Basically, this control is the improved version of the ScriptManager control and designed specially to work with the ASP.NET ACT controls.
UpdatePanelAnimationExtender	Enables you to display animation as the UpdatePanel control is performing an asynchronous postback. This extender applies animation to a very specific situation where custom events need handling before and after an updatable region is refreshed.
ValidatorCalloutExtender	Enables you to display the Validator's error messages by using the balloon-style tool tips of the Windows operating system. This extender enhances the graphics capabilities of the existing the ASP.NET Validator controls.

Installing the ASP.NET AJAX Control Toolkit

The ASP.NET ACT is a shared source of projects created by the joint effort of Microsoft developers and ASP.NET and AJAX community; therefore it is not included within ASP.NET 3.5 Framework. In other words, the Toolkit is community supported and community driven. If you want to avail the extended functionalities of the ASP.NET controls, you need to download and install the ASP.NET ACT manually. The ACT can be downloaded (available free of cost) with lots of additional AJAX controls and components. The Toolkit is continuously updated and a new release of Toolkit is available every couple of months.

The Toolkit is maintained as a project at Microsoft CodePlex. You can get more information and download the latest release of ASP.NET ACT from the following location:

http://www.asp.net/AJAX/AjaxControlToolkit

The CodePlex project allows you to download a compiled `.dll` with the addition of latest controls and extenders. Alternatively, you can also download the source code and project files to compile them on your own system. Both the alternatives require the generated `.dll` file, which should be added to your Toolbox in Visual Studio window.

Installation of the ASP.NET AJAX Control Toolkit is different from general software installation. Perform the following steps to install the ASP.NET AJAX Control Toolkit:

1. Log in with the user who has administrator rights, otherwise during the installation process, a security error will prevent the Toolkit installation.
2. For smooth installation, uninstall the earlier versions of ASP.NET AJAX from your system, otherwise the installation process will terminate.

NOTE

You can uninstall the earlier ASP.NET AJAX versions from the Programs and Features section available under Programs in the Control Panel.

3. Download the compressed file from http://www.asp.net/AJAX/AjaxControlToolkit.
4. Decompress the files to any location on your system and change its name to a desired name, say, `AjaxControlToolkit`.
5. Open the Visual Studio window and right-click the Toolbox window. Then select Add Tab from the context menu, as shown in Figure 32.15:

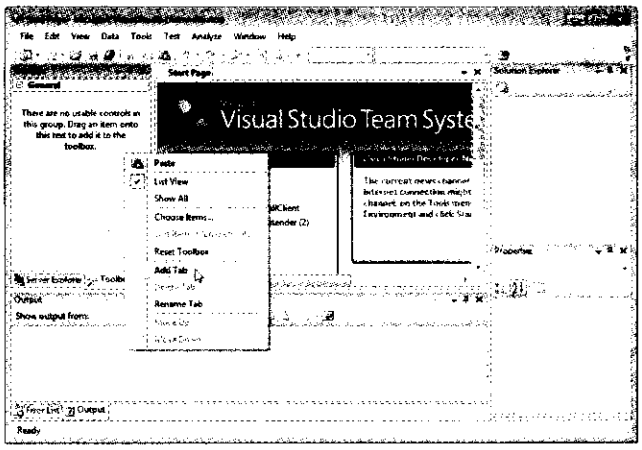


Figure 32.15: Creating Tab in Visual Studio Window

6. Type a tab name, say, `AjaxControlToolkit`, for the newly created tab. Right-click the `AJAXControlToolkit` tab and select Choose Items from the context menu, as shown in Figure 32.16:

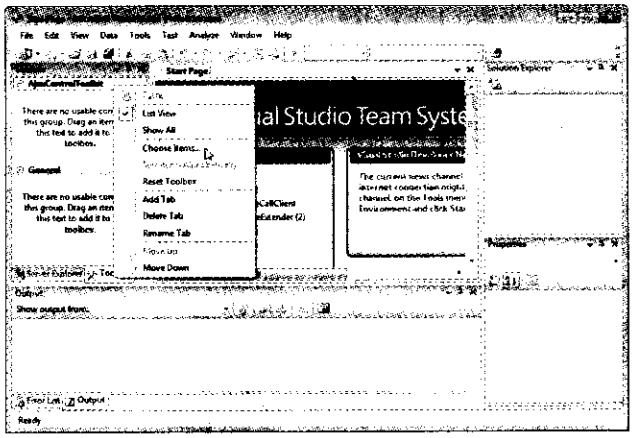


Figure 32.16: Selecting Choose Items Option

The Choose Toolbox Items dialog box opens, as shown in Figure 32.17:

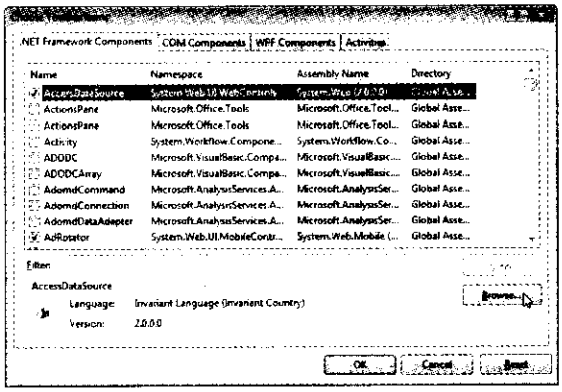


Figure 32.17: Choosing Items for the Tab

7. Click the Browse button, as shown in Figure 32.17. Move to the directory where you have decompressed the AJAX Toolkit files, as shown in Figure 32.18:

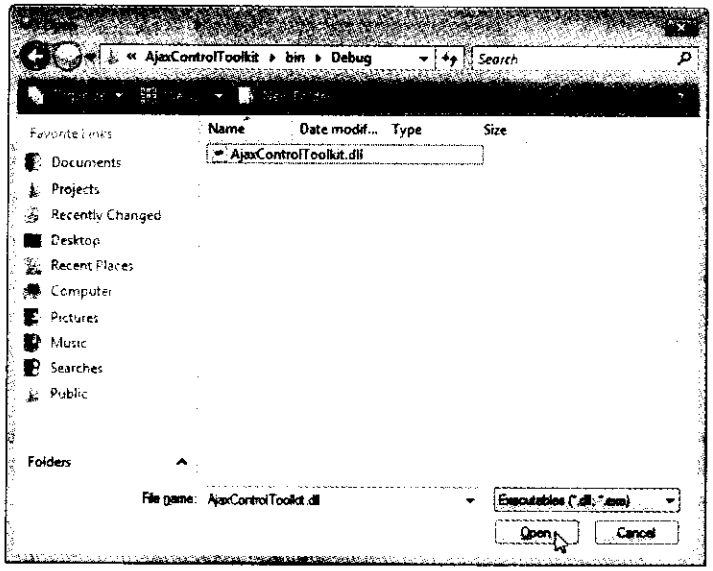


Figure 32.18: Browsing the dll File

8. Select `AjaxControlToolkit.dll` file and click `Open`, as shown in Figure 32.18. The Choose Toolbox Items dialog box opens with the AJAX Toolkit control selected, as shown in Figure 32.19:

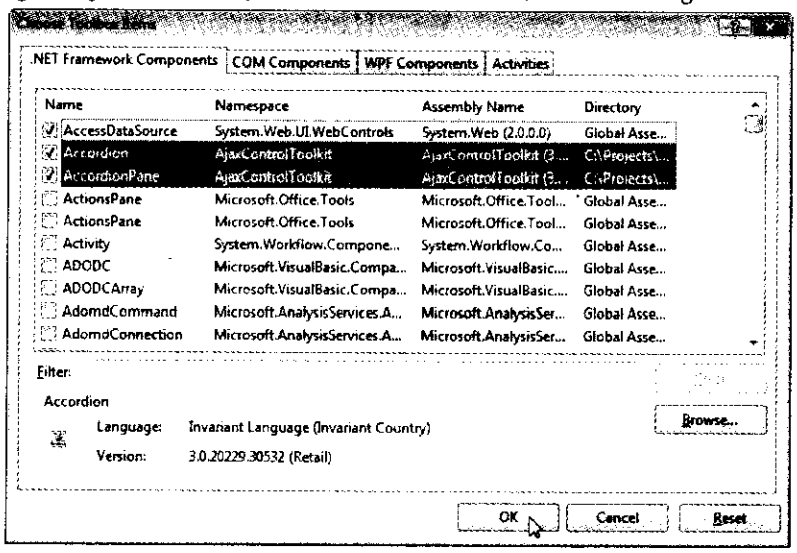


Figure 32.19: Completing the Installation Process

9. Click the `OK` button, as shown in Figure 32.19. The Toolkit controls have been added to the AjaxControlToolkit tab.
10. Create a Web application or a website. Note that, now when you open the Toolbox, a set of all the AjaxControlToolkit controls appear under the AjaxControlToolkit tab, as shown in Figure 32.20:

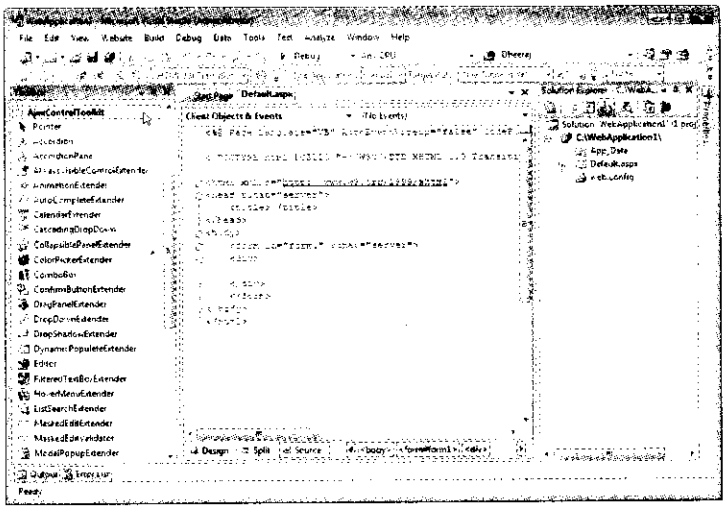


Figure 32.20: Verifying the AJAX Control Toolkit Items

Under the AjaxControlToolkit option, the names of the controls appear and are self-explanatory. This makes it easier to sort out the associativity of the Toolkit's control with the ASP.NET controls.

Note that the Toolkit project is updated regularly and will continue to evolve with the developer's contribution. The controls in this chapter are up to date with the current date of writing, but it is expected that more controls will be added to the Toolkit regularly.

Now, let's start using the Toolkit's extenders and controls in the next section.

Using the ASP.NET AJAX Control Toolkit

After installing ACT, its controls are available for use in Web applications. Now, you will learn to use ACT controls in Web applications.

Using AlwaysVisibleControlExtender

`AlwaysVisibleControlExtender` enables you to fix a given control to one of the page corners so that the control appears to float in the background when the page is scrolled or resized. This extender can target any of the controls that you want to be visible on the page always.

Now, let's create a website, named `AlwaysVisibleControlExtenderVB`. The `AlwaysVisibleControlExtenderVB` website includes a Web page (`Default.aspx`) that contains a lot of text information on the page. The Web page also includes a `Panel` control having a `Label` control and `Image` control inside it. You can find this code of `AlwaysVisibleControlExtenderAppVB` application in the Code\ASP.NET\Chapter 32\`AlwaysVisibleControlExtenderAppVB` folder on the CD. Listing 32.9 shows the code for the `Default.aspx` page of `AlwaysVisibleControlExtenderVB` website:

Listing 32.9: Showing the Code for the `Default.aspx` Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit" TagPrefix="cc1" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">

  <title>AlwaysVisibleControlExtender</title>
```

```
<link href="StyleSheet.css" rel="stylesheet" type="text/css"/>
</head>
<body>
  <form id="form1" runat="server">
      <div>
          <div id="header">

          </div>
      <div id="sidebar">
          <div id="nav">
               
          </div>
      </div>
      <div id="content">
          <p>
           </p>
          <div class="itemContent">
             
          <asp:ScriptManager ID="ScriptManager1" runat="server">
          </asp:ScriptManager>
          <div>

          <asp:Panel ID="Panel1" runat="server" BorderWidth="4" BorderColor="Red"
            HorizontalAlign="Right">
          <div style="height:100px">
          <asp:Label ID="Label1" runat="server" Text="Created by Kogent Solutions
            Inc." Font-Bold="true" Font-Size="Larger" ForeColor="#FF3399"
            BackColor=green Height="300px"></asp:Label>
          </div>

          <asp:Image ID="Image1" runat="server" ImageUrl="~/Garden.jpg"
            Height="275px" Width="300px" Borderwidth="1" />
          </asp:Panel>
          <cc1:AlwaysVisibleControlExtender ID="AlwaysVisibleControlExtender1"
            runat="server" TargetControlID="Panel1" VerticalSide="Top"
            VerticalOffset="10" HorizontalSide="Right" HorizontalOffset="10"
            ScrollEffectDuration="1.0">
          </cc1:AlwaysVisibleControlExtender> <br />
      </div>
      <div style="width:362px">

ASP.NET is a server-side technology for building web applications. Almost all the work
happens on the web server and not the web browser. Whenever you perform an action in
an ASP.NET page-such as clicking a button or sorting a GridView-the entire page must
be posted back to the web server. Any significant action on a page results in a postback.
If you think about it, this is incredibly inefficient. When you perform a postback in an
ASP.NET page, the entire page must be transported across the Internet from browser to
server. Next, the .NET class that corresponds to the page must re-render the entire page
again from scratch. Finally, the finished page must be sent back across the Internet to
the browser. This whole long, slow, agonizing process must occur even if you are updating
a tiny section of the page. <br /><br />

Using a server-side technology such as ASP.NET results in a bad user experience. Every
time a user performs some action on a page, the universe temporarily freezes. Whenever
you perform a postback, the browser locks, the page jumps, and the user must wait
patiently twiddling his thumbs while the page gets reconstructed. All of us have grown
accustomed to this awful user experience. However, we would never design our desktop
applications in the same way.<br /><br />
```

```
ASP.NET is a server-side technology for building web applications. Almost all the work
happens on the web server and not the web browser. Whenever you perform an action in
an ASP.NET page-such as clicking a button or sorting a GridView-the entire page must
be posted back to the web server. Any significant action on a page results in a postback.
If you think about it, this is incredibly inefficient. When you perform a postback in an
ASP.NET page, the entire page must be transported across the Internet from browser to
server. Next, the .NET class that corresponds to the page must re-render the entire page
again from scratch. Finally, the finished page must be sent back across the Internet to
the browser. This whole long, slow, agonizing process must occur even if you are updating
a tiny section of the page.<br /><br />
Using a server-side technology such as ASP.NET results in a bad user experience. Every
time a user performs some action on a page, the universe temporarily freezes. Whenever
you perform a postback, the browser locks, the page jumps, and the user must wait
patiently twiddling his thumbs while the page gets reconstructed. All of us have grown
accustomed to this awful user experience. However, we would never design our desktop
applications in the same way.<br /><br />

ASP.NET is a server-side technology for building web applications. Almost all the work
happens on the web server and not the web browser. Whenever you perform an action in
an ASP.NET page-such as clicking a button or sorting a GridView-the entire page must
be posted back to the web server. Any significant action on a page results in a postback.
If you think about it, this is incredibly inefficient. When you perform a postback in an
ASP.NET page, the entire page must be transported across the Internet from browser to
server. Next, the .NET class that corresponds to the page must re-render the entire page
again from scratch. Finally, the finished page must be sent back across the Internet to
the browser. This whole long, slow, agonizing process must occur even if you are updating
a tiny section of the page.<br /><br />
Using a server-side technology such as ASP.NET results in a bad user experience. Every
time a user performs some action on a page, the universe temporarily freezes. Whenever
you perform a postback, the browser locks, the page jumps, and the user must wait
patiently twiddling his thumbs while the page gets reconstructed. All of us have grown
accustomed to this awful user experience. However, we would never design our desktop
applications in the same way.

        </div>
        </div>
        </div>
        <div id="footer">
            <p class="left">
            All content copyright &copy; Kogent Solutions Inc.</p>
        </div>
        </div>

    </form>

</body>

</html>
```

In Listing 32.9, the `AlwaysVisibleControlExtender` points to the `Panel` control to appear always on the Web page even when the Web page is scrolled or resized. As seen, code of `Default.aspx` file, the `AlwaysVisibleControlExtender` sets its `VerticalSide` and `Horizontalside` property to Top and Right, respectively. This makes the panel appear at the top right corner of the Web browser. Similarly, the `HorizontalOffset` and `VerticalOffset` property is set to 10, which define the distance of panel from horizontal and vertical edges of the browser. The output of the preceding code is shown in Figure 32.21:

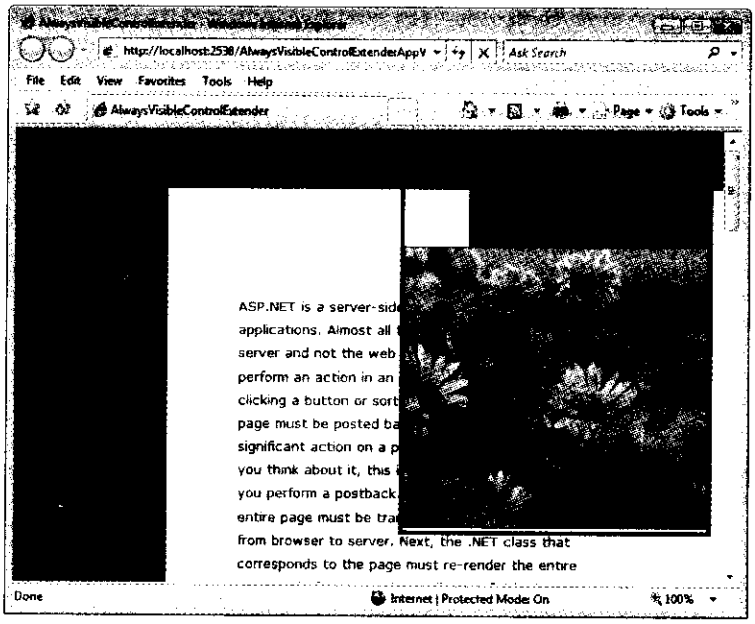


Figure 32.21: Example of the Output of AlwaysVisibleControlExtender

When you scroll the Web page to see the text information, the panel does not scroll with the text information and appears constantly on the Web page.

Summary

In this chapter, you have learned about AJAX in terms of ASP. AJAX calls the server asynchronously and updates a portion of your Web page without refreshing the entire page. AJAX helps the server to improve its performance as it has to process only a small part of page instead of the entire page. Visual Studio 2008 is shipped with built-in AJAX Server controls, used for building AJAX enabled website. These controls are `ScriptManager`, `UpdatePanel`, `Timer`, `UpdateProgress`, and `ScriptManagerProxy`. You have learned to use these controls and their essential properties. After the AJAX Server controls are added to the Web page, standard Web controls (such as `Button`, `Label`, `TextBox` and so on) can be used along with them. If existing list of AJAX Server controls do not meet your requirements, you can create your own controls or customize the existing control's functionality to fulfill your requirement.

There is a broad list of AJAX enabled custom controls, which are not part of Visual Studio 2008; you have to install them separately. This list is named as `AJAX Control Toolkit`. Chapter also discussed the controls of the `AJAX Control Toolkit`.

In the next chapter, we discuss integrating PHP and AJAX.

Quick Revise

Q1. What is ASP.NET AJAX?

Ans. ASP.NET AJAX, mostly called AJAX, is a completely new approach to the Web applications. It is developed by Microsoft as a set of extensions to ASP.NET to implement AJAX functionalities. Using a set of client-side and server-side components, ASP.NET AJAX allows the developers to create Web applications that updates only a specified portion of data without refreshing the whole page. The ASP.NET AJAX uses the AJAX Library that brings Object-oriented programming to JavaScript development for modern browsers and makes it easy to create rich Web applications that communicate with the server using asynchronous postback.

Q2. What do you mean by the term asynchronous postback?

Ans. The asynchronous postback is used in ASP.NET AJAX when the browser sends the callback to the server by creating a new connection; therefore making the callback out-of-band. With this, the entire page does not require to be submitted, so the page lifecycle for a callback skips events associated with rendering page contents on the server. In addition, if the client callback sent to the server is asynchronous, the user can continue working while the request is processing at the server side. When the server responds, the appropriate content on the Web page is updated without refreshing the entire Web page.

Q3. List the essential technologies used in AJAX.

Ans.
- JavaScript
- XmlHttpRequest
- Document Object Model (DOM)
- Extensible HTML (XHTML) and Cascading Style Sheets (CSS)

Q4. What are the different controls of ASP.NET AJAX?

Ans: ASP.NET AJAX includes the following controls:
- ScriptManager
- ScriptManagerProxy
- UpdatePanel
- UpdateProgress
- Timer

Q5. What is the use of the ScriptManager control in AJAX?

Ans: The ScriptManager control is a key component placed at the center of ASP.NET AJAX functionality. Uses of the ScriptManager control are as follows:
- The ScriptManager control coordinates the use of JavaScript for the Microsoft AJAX Library. Each and every Web page that wants to use the AJAX Library must contain a ScriptManager control. The ScriptManager control uses a ScriptManager class to manage AJAX script libraries and script files.
- The ScriptManager control renders the AJAX Library scripts to the browser and supports for partial page rendering, Web service calls, and use of ASP.NET AJAX Client Library.
- The ScriptManager control manages the client-side script of AJAX. AJAX uses JavaScript and therefore needs a mediator to manages the script and restrict a particular version to a browser. The ScriptManager control fulfills this requirement.

Q6. Explain the UpdatePanel control.

Ans. The UpdatePanel control defines portions or regions of a page that can be updated together. As the UpdatePanel refreshes only a selected part of the Web page instead of refreshing the entire page with a postback, you get more flexibility to create rich and client-centric Web applications. Refreshing a selected part of the Web page is referred as partial-page update. You can add one or more UpdatePanel control in the Web page, which automatically participate in partial-page update without custom client script. The UpdatePanel control uses the UpdatePanel class to support for partial-page rendering.

Q7. Why do we use the UpdateProgress control in AJAX?

Ans. The UpdateProgress control is somewhat related to the UpdatePanel control. As asynchronous postback triggers the asynchronous updates, you may keen to know that there is work pending. Otherwise, any time the response is not instantaneous, you feel that if your action has been ignored. Without any visual assistance and acknowledgement of the input that is involved in asynchronous update, the experience can be worse than the screen going blank waiting for a response. The UpdateProgress control helps you to design a more intuitive user interface when a Web page contains one or more UpdatePanel controls for partial-page rendering. Therefore, UpdateProgress control makes you aware of the status information about the partial-page updates in the UpdatePanel control.

Q8. Explain the need of the Timer control in AJAX.

Ans. The need of the Timer control in AJAX can be understood with the help of following points:

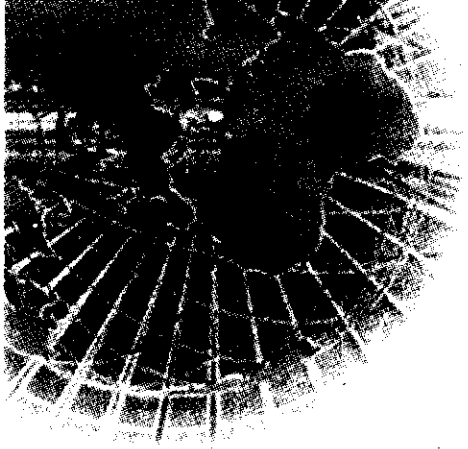
- The Timer control is used with an UpdatePanel control to enable partial-page updates at a defined interval. Alternatively, the Timer control also posts the entire page. The Timer control is mostly used when a periodically partial-page update for one or more UpdatePanel controls is required without refreshing the entire page. Although, you can also use the Timer control to run code on the server every time a Timer control causes a postback and when you want to synchronously post the entire Web page to the Web server at defined intervals.
- The Timer control is a server control that sets a JavaScript component in the Web page, which initiates the postback from the browser just after the time defined in the Interval property is elapsed. The Interval property of the Timer control, defined in milliseconds, specifies how often the postback will occur. Similar to the UpdatePanel control, the Timer control also requires an instance of the ScriptManager control in the Web page. When the Timer control initiates a postback, the Tick event is raised on the server for which you can create an event handler to perform actions when the page is posted to the server. The Tick event occurs when the time specified in the Interval property has elapsed and the page is posted to the server. You can add one or more Timer control on a Web page. This varies with the UpdatePanel that is to be updated. Usually the entire page requires only a single Timer control but you can use multiple Timer control if the UpdatePanel controls are being updated at different intervals.

Q9. Describe AJAX Control Extender Toolkit.

Ans. AJAX Control Toolkit is a set of extenders that are used to extend the functionalities of the ASP.NET controls. The extenders use a block of JavaScript code to add new and enhanced capabilities to the ASP.NET controls. AJAX Control Toolkit is a free download available on the Microsoft site. Before using the extenders, you need to install it on your system.

Q10. Is the AjaxControlToolkit.dll file installed in the Global Assembly Cache?

Ans. No, the AjaxControlToolkit.dll file is not installed in the Global Assembly Cache; it needs to be copied in the Bin folder of your application. However, you may not require to copy the AjaxControlToolkit.dll file explicitly to the Bin folder; instead it gets copied automatically, as soon as you build your website.



33

Integrating PHP and AJAX

So far we learned how JSP and ASP.NET are used as a server-side language to process AJAX request. Besides these two server side languages, there is another language called PHP, which is widely used on server-side. In this chapter, we learn how to handle AJAX request using PHP. This chapter implements the concept of AJAX in PHP based Web applications. Many tools like progress bar, suggested search, internet poll can also be created using AJAX and PHP and integrated in Web applications to make them more responsive. The chapter starts discussing about basic building blocks of PHP, installation of PHP server and making PHP-AJAX enabled pages with using Sajax (AJAX-PHP Framework). Let's start with a brief introduction to PHP.

Introducing PHP

PHP is a server-side scripting language embedded in html. An acronym for PHP is hypertext preprocessor. It is based on C under UNIX operating system and is made by taking the features of Perl and UNIX scripting language. It is set of library routines includes a collection of UNIX system calls and native interfaces to many databases, like Oracle, MySQL, etc. Several organizations use it since it runs on different platforms, like Windows, UNIX, and Linux, etc. and supports various databases MySQL, Oracle, Informix, Sybase, etc.

Building Blocks

PHP consists of different types of variables, arrays, functions, branching and control structures, like if else, do while, for, etc. A PHP file is saved under the extensions `.php`, `.php3`, `.phtml`, `.ph3`, `.ph4`, `.php5`, `.ph5`, etc. A PHP file can have text, HTML tags, and scripts. These scripts are executed on PHP enabled server. PHP scripts are embedded inside HTML page using `<?php and ?>` tags, but some servers use and understand `<? and ?>` tags. Each PHP statement must end with a semicolon.

The code given in Listing 33.1 shows the use of echo statement:

Listing 33.1: Using echo Statement

```
<html>
  <head><title>An echo statement example</title></head>
  <body>
      <h1>An echo statement example</h1>
      <?
          echo "Hi from PHP.";
      ?>
  </body>
</html>
```

An echo statement is used to display text on the browser by PHP. This text is treated as XML by the browser, therefore set the content-type header to text/xml by using the header statement.

Variables

Variables are used to store different types of data. Variables in PHP are similar to those in JavaScript. They can hold numbers, arrays, or objects. Each PHP variable begins with a dollar sign character, `$`. No declaration of variable is required. We can pass several items to the echo statement, with each separated by a comma. To introduce HTML effects, like indentation, `<br>` or `<p >` tags are sent as item. Text strings are assigned to a variable by using following syntax:

```
$var string="Hi from PHP.";
```

Strings

String is a collection of characters. We can perform many operations with strings. Usually '+' is used to concatenate two strings. In PHP, more than one string is concatenated using the dot operator. Table 33.1 lists more string related functions:

Table 33.1: Built-in String functions

`trim`	It eliminates extra spaces from start and end of a string
`substr`	It picks substring from a string

Table 33.1: Built-in String functions	
Function	**Description**
strpos	It locates position of substring in a string
ucfirst	It capitalizes first letter of string
substr_replace	It can change substring with specified string
strtoupper	It can change string to uppercase

Arrays

An array consists of a set of similar elements stored in contiguous locations. We use the `array` statement to create an array in PHP. Here's an example:

```
$arr=array(10,20,30);
```

Arrays elements are accessed by using the syntax $name[index], where name is the name of array and index is the subscript. Here's an example which accesses the first element of array arr:

```
echo $arr[0];
```

You can also access elements in an array by using the text index. Here's another example:

```
$data["temp"] = 98;
```

In this example, "temp" text is used to store the value 98.

Comments

To comment a PHP code, three types of comments are used. The first type of comment starts with `/*` and ends with `*/`. These symbols can contain multiple lines of text. The second and third type of comments start with `//` or `#`, respectively. They are one line comments and these comments end at end of the line.

The code given in Listing 33.2 shows the use of each type of comment:

Listing 33.2: Three Types of Comments

```
<?
/* This is comment of  Type 1*/
//This is comment of type 2
#This is comment of type 3
  echo "Hi from PHP.";
?>
```

Operators

Operators act upon one or two operands perform some operation and generate a result. PHP has a set of operators which is same as that of JavaScript. Precedence order of these operators is same as that in JavaScript. Table 33.2 lists the precedence of different types of operators:

Table 33.2: Operator precedence	
Operators	**Precedence**
Multiplicative	* / %
Additive	+ -
Shift	<< >> >>>
Unary	++expr –expr +expr -expr ~ !
Postfix	expr++ expr--
Relational	< > <= >= instanceof
Equality	== !=
Bitwise	& ^ |

Table 33.2: Operator precedence	
Operators	Precedence
Logical	&& ||
Assignment	= += -= *= /= %= ^=

Control Structures

Control structures are used to control the flow of execution of program. The if else statement is used to do structural programming in PHP.

The code given in Listing 33.3 shows the use of if else statement:

Listing 33.3: Using if else Statement

```
<body>
<h1>Using if else statement</h1>
<?
   $marks=70;
   If($marks<60)
   {
         echo " Student passed in second division.";
   }
   else
   {
         echo "Student passed in first division";
   }
?>
</body>
```

PHP also has loop statements, like for, while, and do while. They work in the same way as in JavaScript, except that the dollar sign is attached to a loop variable.

The code given in Listing 33.4 shows the use of for loop:

Listing 33.4: Using for Loop

```
<body>
   <h1>Using for loop</h1>
   <? for($countter=0;$counter<5;$counter++)
   {
         echo " This line is displayed five times.<br>";
   }
   ?>
</body>
```

Other loops have similar syntax as that of JavaScript. PHP has another type of loop statement, the `foreach` loop. It allows you to iterate on arrays and multiple item objects. You do not need to know the number of items in an array in advance. Value of the current element in each iteration is assigned to `$value` and the array pointer is automatically advanced by one.

The code given in Listing 33.5shows the use of `foreach` loop to generate XML document:

Listing 33.5: Using foreach Loop

```
<?
header('Content-Type: text/xml');
$data = arr('Welcome', 'in', 'XML.');
echo '<?xml version="1.0" ?>';
echo '<document>';
foreach ($data as $value)
{
   echo '<data>';
   echo $value;
```

```
    echo '</data>';
}
echo '</document>';
?>
```

Functions

PHP's strength is that it has a lot of built-in functions. They exceed in the number greater than 700. The syntax of user-defined functions in PHP is similar to functions in JavaScript.

The code given in Listing 33.6shows the use of function:

Listing 33.6: User-defined Function

```
<html>
<body>
<?php
function displayHello()
  {
  echo "Hello";
  }
displayHello();
?>
</body>
</html>
```

Installing PHP Server

To run any simple PHP Web application, you need a PHP enabled server. Servers supporting PHP are Apache server and Microsoft Internet Information Server. Check whether your server supports php. If no, you need to install PHP and its server. Install PHP, Apache Server and MySQL database, if required, in three separate steps or visit hyperlink `http://prdownloads.sourceforge.net/appserv/appserv-win32-2.5.9.exe?download` to download windows installer which installs three in one step. Type the server name `localhost` and administrator the email address `admin@localhost`. Mention the password in password dialog box. To test whether the installation is successful, type `http://localhost` and Figure 33.1 appears:

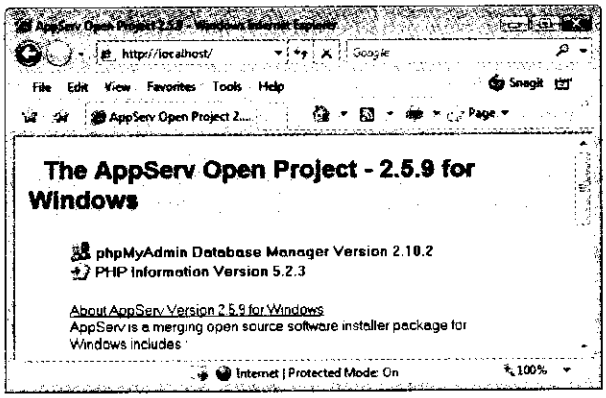


Figure 33.1: Showing Welcome Page of Apache Server

Let's now see how data is sent to a server using PHP and fetched in PHP.

Sending Data to Server

When a web page is submitted, two attributes are used to send data to the server. These attributes are `action` and `method`. The URL of PHP page is assigned to `action` attribute. If we do not use this attribute, the default value is URL of the current PHP document. The method attribute takes two values—`GET` and `POST`. The default value is `GET`.

The code given in Listing 33.7 shows the name.html web page:

Listing 33.7: name.html page

```
<html>
<head><title> Sending data to server</title></head>
<body><center>
<h2>
Sending data using text fields
</h2>
<form method="post" action="displayname.php">
Enter your  name:
<input name="name" type="text">
<br><br>
<input type="submit" value="Submit">
</form></center>
</body>
</html>
```

Listing 33.7 is a simple HTML form having a text field and submit button. When this form is submitted, the control transfers to displayname.php page.

The code given in Listing 33.8 shows the use of $REQUEST array in example:

Listing 33.8: displayname.php

```
<html>
<head><title>Reading data</title>
</head>
<body><center>
<h1>Reading data from text fields in PHP</h1>
Your name is
<?
echo $_REQUEST["name"];
?></center>
</body>
</html>
```

In Listing 33.8, the displayname.php page extracts data from the text field name. PHP $_GET or $_POST array is used to get data, e.g. name, from name.html page depending upon the method used. The name.html file uses the POST method. Therefore, the syntax for accessing text field name is $_POST ["name"]. But, PHP provides another alternative for accessing GET or POST data, which is $_REQUEST array. The `$_REQUEST` is another PHP array, which accesses data without considering GET or POST method.

Let us discuss in brief about AJAX-PHP Frameworks, which help in the creation of AJAX-based PHP Web application. We will talk about syntactic details of only one AJAX-PHP Framework, Sajax, in the next section.

AJAX-PHP Frameworks

All PHP Frameworks provide different ways to affect the client view in both template-based view rendering technologies (like JSP, ASP, and RHTML) and server-side technologies with self rendering capabilities (like ASP.NET and JSF). These frameworks now support the creation of JavaScript for clients to have AJAX effects. This functionality comes in the form of server-side packages. These server-side AJAX packages automatically create JavaScript code for you. Alternative packages also exist in server-side frameworks based on other server-side languages, such as JSP. AJAX integration with other technologies in these frameworks can be performed in many ways, such as using visual tools, using tag libraries, and ORB remoting.

Many PHP Frameworks are available, like AJAXCore, Cake PHP, Xajax, Sajax, XOAD, Zephyr, Feather AJAX 1.1, and Tigermouse to integrate AJAX with PHP. All these frameworks make packaging and deployment of Web applications easier. These frameworks support Model, View, and Controller (MVC) architecture and are based on certain conventions. They free you from repeatedly writing the same code and common functions that are used for different projects.

The Sajax is an AJAX based Framework, which generates AJAX-enabled JavaScript from many server-side languages, like ASP, ColdFusion, Io, Lua, Perl, PHP, Python, and Ruby on server-side. This is a framework that lets you bind user interface to server-side methods. Binding happens by using JavaScript methods which call the Sajax bridge for executing server-side code. This framework uses the Object Remoting technique. In this technique, the object brokers enable remote exchanges and the client-side methods are tied to the server-side objects in a manner that the user feels that there is no network call or round trip between the different entities. In fact, there are network round trips involved and messages are sent via service oriented frameworks. As Sajax exposes its functionality via object broker, Sajax-based application is efficient at development time and slower at runtime.

NOTE

Go to www.modernmethod.com/sajax to download Sajax Framework.

Now let's understand the working of Sajax Framework using a simple example. This example generates a JavaScript function from PHP function on the server-side that manipulates data and returns result to another JavaScript function on client side. The home page, say add.php, of this example has three text fields and one submit button. The user after entering two values in the first two text fields, clicks on Submit button, the result will be displayed in third text field.

For Sajax functionality, you have to include Sajax.php in add.php and need to set up Sajax using sajax_init() method. This page also contains a user-defined function, add, which adds values entered by the user. To access this function in JavaScript by another name, we need to export this method. The export method enables you to access add PHP method using name x_add. Listing 33.9 shows the part of add.php page:

Listing 33.9: Portion of add.php Page

```
<?
require("Sajax.php");
function add($op1, $op2)
{
  return $op1 + $op2;
}
sajax_init();
sajax_export("add");
sajax_handle_client_request();
?>
```

In the end of Listing 33.9, handle_client_request method connects the add function with Sajax and generates JavaScript. You can embed generated JavaScript code in web page using the sajax_show_javascript method. It is done using following syntax:

```
<script>
<?
sajax_show_javascript();
?>
</script>
```

When a user clicks on Submit button, the function corresponding to onclick event invokes x_add generated JavaScript function which in turn calls add function residing on server-side. Listing 33.10 shows the complete code add.php page including preceding code-snippet:

Listing 33.10: JavaScript Part of add.php Page to Perform this Operation

```
<script>
<?
sajax_show_javascript();
?>
function show_results(result) {
  document.getElementById("result").value = result;
}
function do_add()
{
```

```
    var op1, op2;
    op1 = document.getElementById("op1").value;
    op2 = document.getElementById("op2").value;
    x_add(op1, op2, show_results);
  }
  </script>
```

The x_add function takes three arguments first value, second value, and a function name which will display the result of addition.

By now you must be aware that AJAX is a combination of technologies—JavaScript fetches data from the server with `XMLHttpRequest` object and DOM puts the data into the website. This entire process happens asynchronously, without the need to reload the next page. The XMLHttpRequest object becomes very useful to enhance the user experience when used with server-sided script PHP. All operations, like validation, auto suggest, progress bar, live search, poll, dynamically filling, auto-complete, and much more can be performed by using PHP and AJAX.

After exploring the concept of PHP, its relation with AJAX and AJAX PHP Frameworks, let's create Web applications that implement these concepts, in the following section.

Performing Validation

Validation may involve validating a simple integer, or email id, or phone number and social security number, etc. Let's create a Validation application that mimics UI for choosing username for login into any chat room group. The index page for the application displays the text field for you to enter the username. It has JavaScript code to create the `XMLHttpRequest` object. The `validate` method sends request to validate the `.php` page with parameter name. When the status of response is ok, the `<div>` element, having `id res`, is populated with text response received from server. The `abort` method exits from the previous request when a new request comes in.

The code given in Listing 33.11 shows the index page for the application (you can find the ajax.html file in the Code/AJAX/Chapter 33/validation folder on the CD):

Listing 33.11: The ajax.html File

```
<script type="text/javascript">
var http = false;
if(navigator.appName == "Microsoft Internet Explorer") {
  http = new ActiveXObject("Microsoft.XMLHTTP");
}
else
{
  http = new XMLHttpRequest();
}
function validate(name)
{
  http.abort();
  http.open("GET", "validate.php?name=" + name, true);
  http.onreadystatechange=function()
  {
        if(http.readyState == 4)
        {
              document.getElementById('res').innerHTML = http.responseText;
        }
  }
  http.send(null);
}
</script>
<h1>Type  username to choose:</h1>
<form>
  <input type="text" name="name" onkeyup="validate(this.value)" />
```

```
    <div id="res"></div>
</form>
```

The PHP script `validate.php` checks whether the username entered in the form is valid. The file has a `validate` function in which the username entered by a user is passed as argument. The `validate` function checks whether the user has entered any username, matches it with the already existing usernames to enforce the user to choose different usernames, and checks the minimum length of username. The built-in function, `trim`, eliminates extra white spaces from the entered username.

The code given in Listing 33.12 shows the PHP page (you can find the validate.php file in the Code/AJAX/Chapter 33/validation folder on the CD):

Listing 33.12: The validate.php File

```
<?php
function validate($name)
{
  if( $name== ' ')
  {
        return 'Please enter any username';
  }
  if(strlen($name) < 3)
  {
        return "Username is too short";
  }
  switch($name)
  {
        case 'tom':
        case 'jerry':
        case 'pop':
        case 'caren':
        return "Username already exists.";
  }
  return "Username is valid.";
}
echo validate(trim($_REQUEST["name"]));
?>
```

Copy these two files into the `validation` folder under the `www` root directory of the server. Type `http://localhost/validation/ajax.html` in the address bar of Internet Explorer. In Figure 33.2, when the user enters a string (less than three characters), the message, Username is too short is displayed, as shown in Figure 33.2:

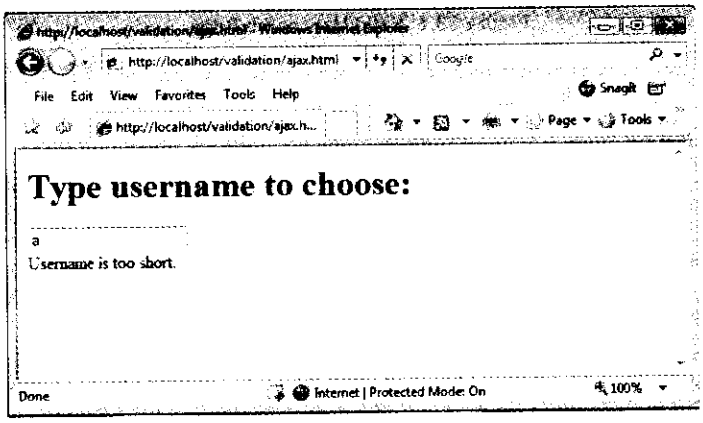


Figure 33.2: Checking Small Username

When the user enters tom, which already exists in one of the case expression of switch statement in Listing 33.12, the message Username already exists is displayed, as shown in Figure 33.3:

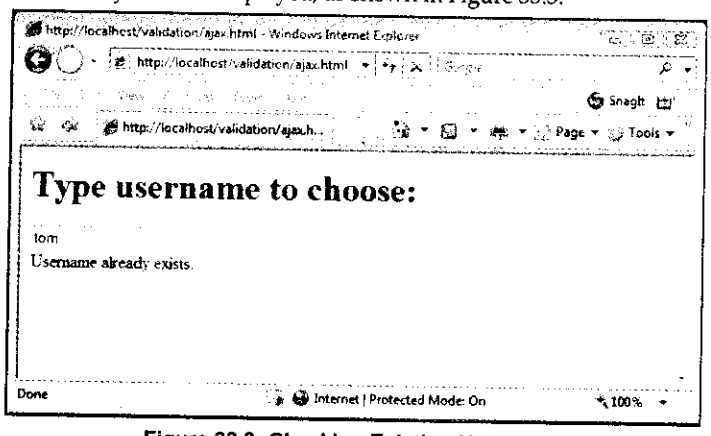


Figure 33.3: Checking Existing Username

When the user enters username Steveo whose length is greater than 3 characters and also it does not match any of names in case expressions of switch statement of Listing 33.12, the message Username is valid is displayed, as shown in Figure 33.4:

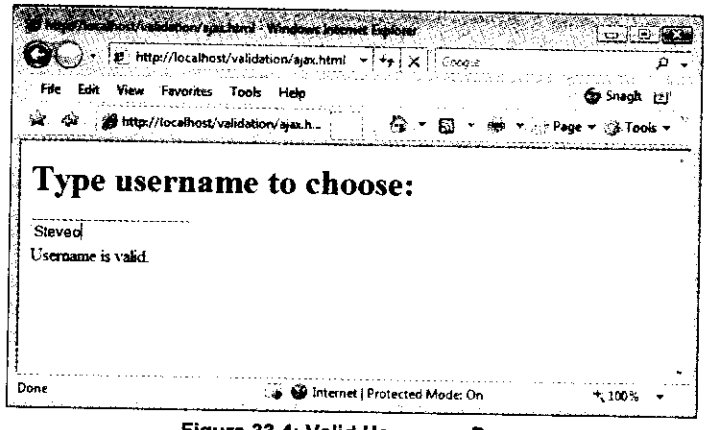


Figure 33.4: Valid Username Page

Let's make an application which asks people or users about their opinions on one question and displays final results in graphical manner.

Polling Application

Polling questions are asked from people to take a survey about some parameters, like birth rate, GDP, etc. They are either in the form of yes/no poll or multiple choice polls. A simple survey has three parts—one part to display a sample question for visitors; another part to let visitors vote for question and; lastly to allow visitors to view results. All three parts require two web pages to be loaded in non-AJAX based application. In AJAX-based application, there is no need to load pages. Only one home page of application refreshes in background. This application demonstrates the real online yes/no poll.

The `index.html` page includes `ajax.js` JavaScript file and consists of simple HTML forms. The form contains two radio buttons. Values 0 and 1 associated with these buttons are used to depict, if the user clicked yes or no on the server-side. Clicking on any of the two radio buttons calls the `sendVote` method with a value passed as parameter.

The code given in Listing 33.13shows code for the `index.html` page (you can find this file in the Code/AJAX/Chapter 33/poll folder on the CD):

Listing 33.13: The index.html File

```
<html>
<head>
<script src="ajax.js"></script>
</head>
<body><div id="poll">
<h2>Do you believe in existence of dinosaurs? </h2><form>
Yes:
<input type="radio" name="vote"
value="0" onclick="sendVote(this.value)">
<br>No:
<input type="radio" name="vote"
value="1" onclick="sendVote(this.value)">
</form>
</div></body>
</html>
```

The `GetXmlHttpObject` retrieves the object of `XMLHttpObject`. If this function fails to retrieve it, the message "Browser does not support HTTP request" is displayed. This makes the URL to request the `vote.php` file. This URL has `vote` and `sid` parameters. The vote is initialized to get the value of the clicked radio button and the `sid` parameter is initialized to get the random number, which is generated by `Math.random` built-in function. If a random number is not attached to a url, the server will display the same results stored in its buffer. At last, it sends a request to `vote.php` page with these parameters. When the server completes its response, the `stateChanged` method is called. It updates the `<div>` poll with the text of response.

The code given in Listing 33.14shows code for the ajax.js file (you can find this file in the Code/AJAX/Chapter 33/poll folder on the CD):

Listing 33.14: The ajax.js File

```
var xmlHttp
function sendVote(int)
{
  xmlHttp=GetXmlHttpObject()
  if (xmlHttp==null)
  {
        alert ("Browser does not support HTTP Request")
        return
  }
  var url="vote.php"
  url=url+"?vote="+int
  url=url+"&sid="+Math.random()
  xmlHttp.onreadystatechange=stateChanged
  xmlHttp.open("GET",url,true)
  xmlHttp.send(null)
}
function stateChanged()
{
  if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
  {
        document.getElementById("poll").
        innerHTML=xmlHttp.responseText;
  }
}
function GetXmlHttpObject()
{
  var objXMLHttp=null
  if (window.XMLHttpRequest)
```

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

The `result.txt` file stores data in the format shown in Listing 33.15. The first number is the count of 'Yes' votes, while the second is the count of 'No' votes. Both numbers are separated by string `||`.

The code given in Listing 33.15 shows the code for result.txt (you can find this file in the Code/AJAX/Chapter 33/poll folder on the CD):

Listing 33.15: The result.txt File

```
0||0
```

The `vote.php` file retrieves the value of vote request parameter. The file function reads the whole file into the content variable in the form of an array. The explode function splits the array by string `||`. It inserts this string after the contents of the first element of content array. The yes variable stores the number of Yeses on poll and no variable stores the number of Nos on poll. Initially, the `yes` variable is initialized to the first element of `array` variable and `no` variable is initialized to the second element of `array` variable. Upon clicking any of the radio buttons each time, the corresponding yes or no variable is incremented by one. Then the `result.txt` file is opened in write mode using fopen method and inserts votes to `result.txt` file using fputs method. Relative ratio of yes votes is calculated by using the number of yes votes divided by the total votes formula. Similarly, ratios of no votes are calculated. The round function rounds the ratio to the first two decimal places after decimal point. The percentage in each case is calculated by multiplying 100. The percentage of yes or no votes is shown by image, `img`. The `width` of the image is controlled according to the percentage.

The code given in Listing 33.16 shows the code for the `vote.php file` (you can find this file in the Code/AJAX/Chapter 33/poll folder on the CD):

Listing 33.16: The vote.php File

```
<?php
$vote = $_REQUEST['vote'];
$filename = "result.txt";
$content = file($filename);
$array = explode("||", $content[0]);
$yes = $array[0];
$no = $array[1];if ($vote == 0)
{
  $yes = $yes + 1;
}
if ($vote == 1)
{
  $no = $no + 1;
}
//insert votes to txt file
$insertvote = $yes."||".$no;
$fp = fopen($filename,"w");
fputs($fp,$insertvote);
fclose($fp);
?><h2>Result:</h2>
<table>
<tr>
<td>Yes:</td>
<td>
<img src="image.gif"
```

```
width='<?php echo(100*round($yes/($no+$yes),2)); ?>'
height='20'>
<?php echo(100*round($yes/($no+$yes),2)); ?>%
</td>
</tr>
<tr>
<td>No:</td>
<td>
<img src="image.gif"
width='<?php echo(100*round($no/($no+$yes),2)); ?>'
height='20'>
<?php echo(100*round($no/($no+$yes),2)); ?>%
</td>
</tr>
</table>
```

All files are packaged under `poll` directory in the `www` root directory of Apache Server. Type `http://localhost/poll/index.html` at the address bar of the browser and press Enter key. Figure 33.5 appears:

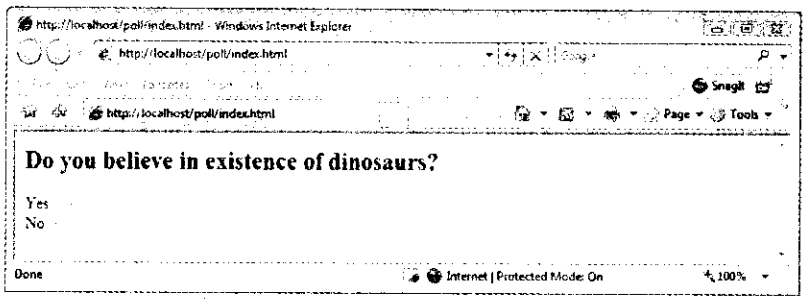


Figure 33.5: Home Page of Poll Application

Suppose we voted four times to Yes option and four times to No option. In that case, the `result.txt` file contains 4||4 upto that instant. In case of Internet polling, this means that total eight users have entered their votes. Clicking on Yes button will display the results, as shown in Figure 33.6:

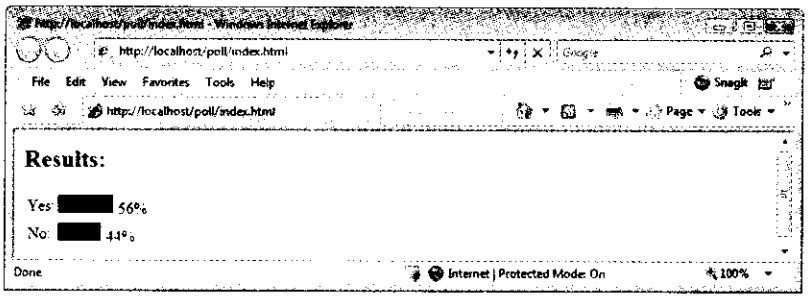


Figure 33.6: Poll Results

Let us make a Suggestions application which suggests words automatically to the user having similar pattern to a word that the user is currently typing.

Suggestions Application

Google Suggest is very large and a good example of suggestions Web applications. In our application, we do not connect to a database. We take a set of values about some specific field and provide user suggestions from that list. This application gives user suggestions about the related words when the user types.

The `index.html` file includes `ajax.js` JavaScript file. It consists of a simple form, which prompts the user to enter some text string. When the user releases the key in input field, the `sendString` method of `ajax.js` file is called. The `<span>` tag is later populated with response from server.

The code given in Listing 33.17 shows the code for `index.html` file (you can find this file in the Code/AJAX/Chapter 33/suggest folder on the CD):

Listing 33.17: The index.html File

```
<html>
<head>
<script src="ajax.js"></script>
</head><body><form>
Type any String
<input type="text" id="name"
onkeyup="sendString(this.value)">
</form><p>Suggested words are : <span id="sug"></span></p> </body>
</html>
```

In Listing 33.18, the `GetXmlHttpObject` retrieves the object of `XMLHttpObject`. If your browser is not one of Opera, Safari, and FireFox, the code in catch statement creates `XMLHttpObject` for Internet Explorer. If this function fails to get `XMLHttpObject`, the message 'Browser does not support HTTP request' is displayed. It makes the URL to request `suggest.php` file. This URL has a parameter `q`. This parameter is initialized to the value of textbox. At last, it sends a request to `suggest.php` page along with this parameter. When the server completes its response, the `stateChanged` method is called. It updates the `<span> sug` element with text of response.

The code given in Listing 33.18 shows the code for `ajax.js` file (you can find this file in the Code/AJAX/Chapter 33/suggest folder on the CD):

Listing 33.18: The ajax.js File

```
var xmlHttp
function sendString(str)
{
  if (str.length==0)
  {
        document.getElementById("sug").innerHTML=""
        return
  }
  xmlHttp=GetXmlHttpObject()
  if (xmlHttp==null)
  {
        alert ("Browser does not support HTTP Request")
        return
  }
  var url="suggest.php"
  url=url+"?q="+str;
  xmlHttp.onreadystatechange=stateChanged
  xmlHttp.open("GET",url,true)
  xmlHttp.send(null)
}

function stateChanged()
{
  if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
  {
        document.getElementById("sug").innerHTML=xmlHttp.responseText
  }
}
function GetXmlHttpObject()
{
```

```
    var xmlHttp=null;
    try
    {
          // Firefox, Opera 8.0+, Safari
          xmlHttp=new XMLHttpRequest();
    }
    catch (e)
    {
          // Internet Explorer
          try
          {
                xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
          }
          catch (e)
          {
                xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
          }
    }
    return xmlHttp;
}
```

In the server page `suggest.php`, an array `a` is initialized with words. When the user enters any text in textbox, this page is requested with text as request parameter. It finds each word which matches the text entered by the user and appends these words to string hint. If no matching word is found, the response string is set to no suggestion.

The code given in Listing 33.19 shows the code for `suggest.php` page (you can find this file in the Code/AJAX/Chapter 33/suggest folder on the CD):

Listing 33.19: The suggest.php File

```
<?php
// Fill up array with names
$a[]="advice";
$a[]="become";
$a[]="commit";
$a[]="dictionary";
$a[]="endeavour";
$a[]="feel";
$a[]="gear";
$a[]="home";
$a[]="intend";
$a[]="jump";
$a[]="know";
$a[]="liaise";
$a[]="near";
$a[]="off";
$a[]="provide";
$a[]="amend";
$a[]="retain";
$a[]="comply";
$a[]="despite";
$a[]="emhance";
$a[]="effect";
$a[]="sustain";
$a[]="tools";
$a[]="undo";
$a[]="violet";
$a[]="leap";
$a[]="enhance";
$a[]="emphasize";
```

```
$a[]="week";
$a[]="year";//get the q parameter from URL
$q=$_GET["q"];//lookup all hints from array if length of q>0
if (strlen($q) > 0)
{
  $hint="";
  for($i=0; $i<count($a); $i++)
  {
        if (strtolower($q)==strtolower(substr($a[$i],0,strlen($q))))
        {
              if ($hint=="")
              {
                    $hint=$a[$i];
              }
              else
              {
                    $hint=$hint." , ".$a[$i];
              }
        }
  }
}
//Set output to "no suggestion" if no hint were found
//or to the correct values
if ($hint == "")
{
  $response="no suggestion";
}
else
{
  $response=$hint;
}
//output the response
echo $response;
?>
```

All files are packaged under the `suggest` directory in `www` root directory of Apache Server. Type `http://localhost/suggest/index.html` in the address bar of browser. When the user enters a character e, all the suggested words starting from e are displayed, as shown in Figure 33.7:

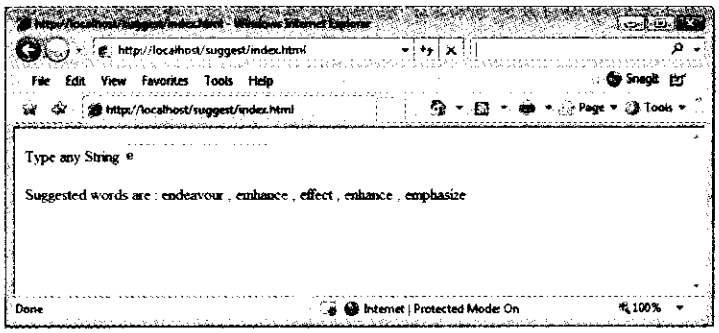


Figure 33.7: Suggestions List

Handling XML Data using PHP and AJAX

Generally, XML files are used to collect or represent data with your own defined tags. In this section, we'll explain how the web page gets data from an XML file by using AJAX. Here we are taking a sample XML file to interact with. The index.html file includes `selectbook.js` JavaScript file and has a simple HTML form. This form displays a list box to the user to select a book title and when the user changes the value in the list box, the

`sendString` method of `ajax.js` file is called. The `<div>` tag `res`, is later populated with the response from server.

The code given in Listing 33.20 shows the code for index.html (you can find this file in the Code/AJAX/Chapter 33/xml folder on the CD):

Listing 33.20: The index.html File

```
<html>
<head>
<script src="selectbook.js"></script>
</head><body><form>
List of Book Titles:
<select name="titles" onchange="sendTitle(this.value)">
<option>Select a Title</option>
<option value="BookTitle1">BookTitle1</option>
<option value="BookTitle2">BookTitle2</option>
<option value="BookTitle3">BookTitle3</option>
<option value="BookTitle4">BookTitle4</option>
</select>
</form><p>
<div id="res"><b>Book Details will be given here.</b></div>
</p></body>
</html>
```

This sample XML file, we used here is `bookstore.xml`. It consists of a collection of books and each `<Book>` element has `Title`, `Author`, `Year`, and `Price` sub elements.

The code given in Listing 33.21 shows the sample code for XML file (you can find this file in the Code/AJAX/Chapter 33/xml folder on the CD):

Listing 33.21: The bookstore.xml File

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Bookstore>
<Book>
<Title >BookTitle1</Title>
<Author>AuthorName1</Author>
<Year>2005</Year>
<Price>35.00</Price>
</Book>
<Book>
<Title>BookTitle2</Title>
<Author>AuthorName2</Author>
<Year>2007</Year>
<Price>30.00</Price>
</Book>
<Book>
<Title>BookTitle3</Title>
<Author>AuthorName3</Author>
<Year>2006</Year>
<Price>50.00</Price>
</Book>
<Book>
<Title>BookTitle4</Title>
<Author>AuthorName4</Author>
<Year>2005</Year>
<Price>40.00</Price>
</Book>
</Bookstore>
```

The `GetXmlHttpObject` method has the same functionality as earlier. It makes the URL to request `getbook.php` file. This URL has a parameter `q`. This parameter is initialized to the selected value of list box. At

last, it sends the request to `getbook.php` page along with this parameter. When the server completes its response, the `stateChanged` method is called. It updates the `<span> res` element with the text of response.

The code given in Listing 33.22 shows the code for `selectbook.js` file (you can find this file in the Code/AJAX/Chapter 33/xml folder on the CD):

Listing 33.22: The selectbook.js File

```
var xmlHttp
function sendTitle(str)
{
  xmlHttp=GetXmlHttpObject()
  if (xmlHttp==null)
  {
        alert ("Browser does not support HTTP Request")
        return
  }
  var url="getbook.php"
  url=url+"?q="+str;
  xmlHttp.onreadystatechange=stateChanged
  xmlHttp.open("GET",url,true)
  xmlHttp.send(null)
}
function stateChanged()
{
  if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
  {
        document.getElementById("res").innerHTML=xmlHttp.responseText
  }
}
function GetXmlHttpObject()
{
  var xmlHttp=null;
  try
  {
        // Firefox, Opera 8.0+, Safari
        xmlHttp=new XMLHttpRequest();
  }
  catch (e)
  {
        // Internet Explorer
        try
        {
              xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
        }
        catch (e)
        {
              xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
        }
  }
  return xmlHttp;
}
```

When the user selects an option, the server page `getbook.php` creates the XML `DOMDocument` object and loads `bookstore.xml` file. All the `Title` elements are iterated to search a title matching the title sent from the HTML form. This way the book containing the correct title is found.

The code given in Listing 33.23 shows the code for `getbook.php` file (you can find this file in the Code/AJAX/Chapter 33/xml folder on the CD):

Listing 33.23: The getbook.php File

```
<?php
$q=$_GET["q"];$xmlDoc = new DOMDocument();
$xmlDoc->load("bookstore.xml");$x=$xmlDoc->getElementsByTagName('Title');
for ($i=0; $i<=$x->length-1; $i++)
{
  //Process only element nodes
  if ($x->item($i)->nodeType==1)
  {
        if ($x->item($i)->childNodes->item(0)->nodeValue == $q)
        {
              $y=($x->item($i)->parentNode);
        }
  }
}
$book=($y->childNodes);for ($i=0;$i<$book->length;$i++)
{
  //Process only element nodes
  if ($book->item($i)->nodeType==1)
  {
        echo($book->item($i)->nodeName);
        echo(": ");
        echo($book->item($i)->childNodes->item(0)->nodeValue);
        echo("<br />");
  }
}
?>
```

All files are packaged under xml directory in www root directory of Apache Server. Type `http://localhost/xml/index.html` in the address bar of browser and press enter. Instantly a screen, as shown in Figure 33.8, gets displayed:

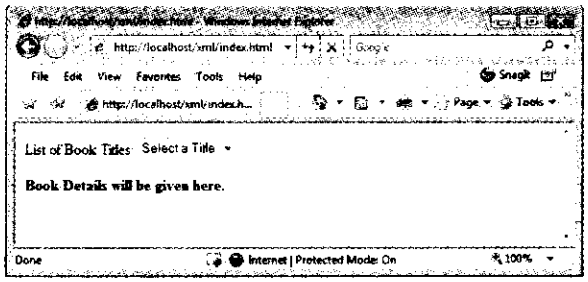


Figure 33.8: Displaying index.html Page

As the user selects the title 'BookTitle2' from the list box, the corresponding book details including title, author name, year of publishing, and price are displayed, as shown in Figure 33.9:

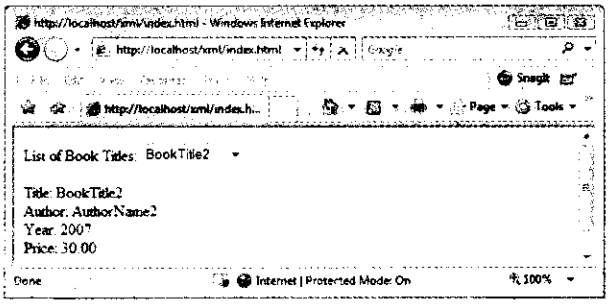


Figure 33.9: Details of Book having Title "BookTitle2"

In all the previous applications, we had not made any interaction with the database. Let's make an AJAX application, which fetches the required data from the database and displays it on the browser.

Fetching Records from Database using AJAX

Some tasks, like database related tasks, are not easily achieved by using JavaScript. The database storage and retrieval tasks can be accomplished by using a combination of MySQL and PHP. This combination results in better processing also. You do not need to install MYSQL database separately, as it is already installed along with the installation of PHP server, which is AppServer.

The database we are using is employees and consists of one table, `emprecord` (shown in Table 33.3):

Table 33.3: emprecord table

Id	Name	Age	City	Designation
1	Gaurav Sachdeva	35	Tirupati	Secretary
2	Sukhjeet Kumar	35	Ranchi	Chairman
3	Gurjeet Narang	41	Gurgaon	Accountant
4	Parul Mittal	30	Manglore	Graphics Designer
5	Ashish Mahindroo	32	Nagpur	Manager

We are creating an example which fetches the record of a particular employee from database table, emp_record. This example starts from index.html page. This page provides a combo box where you can select the name of employee whose record you wish to see.

The `index.html` file includes `ajax.js` JavaScript file and has a simple HTML form. The form consists of a combo box, which contains employees' names and ids that exist in database table emp_record. This combo box shows various employee names to select one name. When the user selects one employee name from the combo box, the `sendEmpID` method of `ajax.js` file is called. The `<div>` tag `emp` is later populated with response from server.

The code given in Listing 33.24 shows the code for `index.html` file (you can find this file in the Code/AJAX/Chapter 33/database folder on the CD):

Listing 33.24: The index.html File

```
<html>
<head>
<script src="ajax.js"></script>
</head>
<body><form>
Employees List:
<select name="names" onchange="sendEmpID(this.value)">
<option>Select Name</option>
<option value="1">Gaurav Sachdeva</option>
<option value="2">Sukhjeet Kumar</option>
<option value="3">Gurjeet Narang</option>
<option value="4">Parul Mittal</option>
<option value="5">Ashish Mahindroo</option>
</select>
</form><p>
<div id="emp"><b>Employee info will be listed here.</b></div>
</p></body>
</html>
```

In Listing 33.25, the `sendEmpID` method invokes the `GetXmlHttpObject` method. The `GetXmlHttpObject` method makes the asynchronous request to the `getemployee.php` file. The request URL has parameters `q` to store `id` and `sid` to store random numbers in order to avoid the server from taking the cached file. At last, it

sends the request to `getemployee.php` page along with this parameter. When the server completes its response, the `stateChanged` method is called. It updates the `<span> res` element with the text of response.

The code given in Listing 33.25 shows the code for ajax.js file (you can find this file in the Code/AJAX/Chapter 33/database folder on the CD):

Listing 33.25: The ajax.js File

```
var xmlHttp;
function sendEmpID(str)
{
  xmlHttp=GetXmlHttpObject();
  if (xmlHttp==null)
  {
        alert ("Browser does not support HTTP Request")
        return;
  }
  var url="getemployee.php";
  url=url+"?q="+str;
  url=url+"&sid="+Math.random();
  xmlHttp.onreadystatechange=stateChanged;
  xmlHttp.open("GET",url,true);
  xmlHttp.send(null); }
function stateChanged()
{
  if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
  {
        document.getElementById("emp").innerHTML=xmlHttp.responseText;
  }
}
function GetXmlHttpObject()
{
  var xmlHttp=null;
  try
  {
        // Firefox, Opera 8.0+, Safari
        xmlHttp=new XMLHttpRequest();
  }
  catch (e)
  {
        //Internet Explorer
        try
        {
              xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
        }
        catch (e)
        {
              xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
        }
  }
  return xmlHttp;
}
```

When the user selects the employee name from the list box, it sends id as query parameter to `getemployee.php` page. The server page `getemployee.php` retrieves this parameter. It establishes a connection to MySQL server by using the username equal to 'root' and password equal to 'root'. Change the username and password, according to what you have specified during installation under topic 'Installing PHP server'. If the connection fails, the `die` method displays an error message. Then, it selects the database employees and sends a query on table `emprecord` based on id. The HTML table is created and the fetched values are inserted into corresponding cells of row.

The code given in Listing 33.26 shows the code for `getemployee.php` page (you can find this file in the Code/AJAX/Chapter 33/database folder on the CD):

Listing 33.26: The getemployee.php File

```
<?php
$q=$_GET["q"];
$con = mysql_connect('localhost', 'root', 'root');

if (!$con)
{
  die('Not able to connect: ' . mysql_error());
}
mysql_select_db("employees", $con);
$query="SELECT * FROM emprecord WHERE id = '".$q."'";
$result = mysql_query($query);
echo "<table border='1'>

<tr>
<th>Name</th>
<th>Age</th>
<th>City</th>
<th>Designation</th>
</tr>";

while($row = mysql_fetch_array($result))
{
  echo "<tr>";
  echo "<td>" . $row['name'] . "</td>";
  echo "<td>" . $row['age'] . "</td>";
  echo "<td>" . $row['city'] . "</td>";
  echo "<td>" . $row['designation'] . "</td>";
  echo "</tr>";
}
echo "</table>";
mysql_close($con);
?>
```

All files are packaged under `database` directory in `www` root directory of Apache Server. Type `http://localhost/database/index.html` in the address bar of the browser and press Enter key. Instantly, a screen, similar to Figure 33.10, appears:

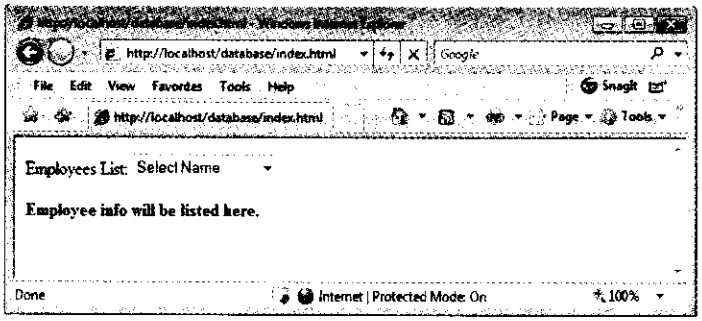


Figure 33.10: Showing the Index.html Page

When we select the employee Ashish Mahindroo from the list, the employee record of this employee is retrieved from the emprecord table by getemployee.php page and displayed in the form of HTML table, as shown in Figure 33.11:

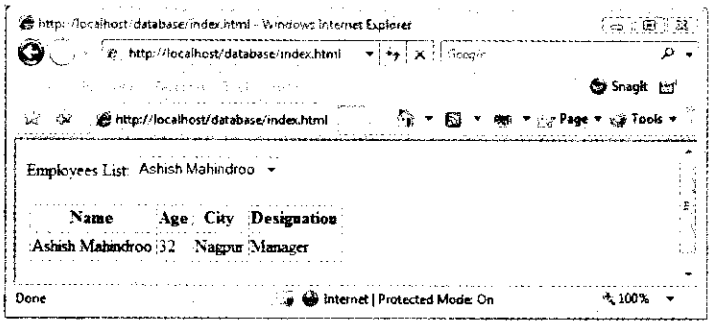


Figure 33.11: Selecting Employee Details

Figure 33.11 shows the employee record consisting of personal details of employee, such as name, age, city, and designation. This application focussed on making a database-driven AJAX application.

Summary

In this chapter, we understood the building blocks of PHP language, such as operators, variables—strings, numbers, arrays, control structures, and functions. We used Apache Server to run PHP-enabled Web applications. Later in the chapter PHP is integrated with AJAX in Web applications, which performed validation, autosuggest, polling, and populating data from MySQL database.

The next chapter will focus on consuming Web Services in AJAX.

Quick Revise

Q1. Define PHP.

Ans: PHP is a server-side scripting language embedded in html. An acronym for PHP is hypertext preprocessor. It is based on C under UNIX operating system and is made by taking the features of Perl and UNIX scripting language. It is set of library routines includes a collection of UNIX system calls and native interfaces to many databases, like Oracle, MySQL, etc. Several organizations use it since it runs on different platforms, like Windows, UNIX, and Linux, etc. and supports various databases MySQL, Oracle, Informix, Sybase, etc.

Q2. List the building blocks of PHP.

Ans: Following are the building blocks of PHP:

- Variables
- Arrays
- Functions
- Control statements

Q3. List the operators in PHP according to their precedence.

Ans: The following table lists the operators in PHP according to their precedence:

Multiplicative	* / %
Additive	+ -
Shift	<< >> >>>
Unary	++expr –expr +expr -expr ~ !
Postfix	expr++ expr--
Relational	< > <= >= instanceof
Equality	== !=
Bitwise	& ^ |

Logical	&& ||
Assignment	= += -= *= /= %= ^=

Q4. Whish are the PHP supported servers?

Ans: Servers supporting PHP are Apache server and Microsoft Internet Information Server

Q5. Define AJAX-PHP Framework.

Ans: Many PHP Frameworks are available, like AJAXCore, Cake PHP, Xajax, Sajax, XOAD, Zephyr, Feather AJAX 1.1, and Tigermouse to integrate AJAX with PHP. All these frameworks make packaging and deployment of Web applications easier. These frameworks support Model, View, and Controller (MVC) architecture and are based on certain conventions. They free you from repeatedly writing the same code and common functions that are used for different projects.

Q6. Give examples of Web applications developed with the help of the AJAX-PHP framework.

Ans: You can develop following types of Web application with the help of the AJAX-PHP framework:

- Performing validation application
- Polling application