Robert Foster

ASP.NET 3.5 AJAX

UNLEASHED

SAMS

ASP.NET 3.5 AJAX Unleashed
Copyright © 2009 by Pearson Education, Inc.

ISBN-13: 978-0-672-32973-9
ISBN-10: 0-672-32973-5

Library of Congress Cataloging-in-Publication Data:

Foster, Robert Hill.
ASP.NET 3.5 Ajax unleashed / Robert Foster.
p. cm.
ISBN 0-672-32973-5
1. Active server pages. 2. Ajax (Web site development technology) 3. Application software–Development. 4. Microsoft .NET. I. Title.
TK5105.8885.A52F67 2008
005.2'76—dc22

2008014763

Printed in the United States of America
First Printing December 2008

Editor-in-Chief
Karen Gettman

Executive Editor
Neil Rowe

Development Editor
Mark Renfrow

Managing Editor
Kristy Hart

Project Editor
Meg Shaw

Copy Editors
Lori Lyons
Karen Annett

Indexer
Erika Millen

Proofreader
Anne Goebel

Technical Editor
Todd Meister

Publishing Coordinator
Cindy Teeters

Book Designer
Gary Adair

Composition
Jake McFarland

Manufacturing Buyer
Dan Uhrig

Introduction

As an ASP.NET web developer in today's market, you need countless skills to distinguish yourself from the next person. One of those skills that you can quickly acquire is developing AJAX-enabled applications. As your users begin to utilize AJAX-enabled sites on the web, such as Live.com and Google Maps, they are beginning to expect the same rich functionality in the applications that you deliver.

Although these applications took many man hours to create, the process is made easier for you as a developer with the AJAX library that's packaged with ASP.NET 3.5. You can save many hours and lines of code by learning and leveraging ASP.NET AJAX to create and deliver a very rich user experience in your applications.

In this book, you learn how to make the most of ASP.NET AJAX by building on the knowledge that you already have as an ASP.NET developer and extend that knowledge so that you can easily create AJAX-enabled applications. The book has been divided into the following chapters:

Chapter 1—Introduction to AJAX Technologies

In this chapter, you learn the fundamentals of AJAX by first understanding the XmlHttpRequest JavaScript object, and then building a simple AJAX library for use in your applications.

Chapter 2—Introduction to ASP.NET AJAX

Chapter 2 introduces you to the controls and capabilities that are available in ASP.NET AJAX and serves as a springboard for technologies that are discussed in future chapters.

Chapter 3—The ScriptManager and ScriptManagerProxy Controls

Chapter 3 discusses the core object of the ASP.NET AJAX library: the ScriptManager control. In this chapter, you learn the capabilities of the ScriptManager control and how you can best utilize it in your AJAX-enabled pages.

Chapter 4—The ASP.NET AJAX Client API

In Chapter 4, you learn how to make the most of the ASP.NET AJAX client-side objects that are made available via the ScriptManager control.

Chapter 5—The UpdatePanel and Timer Controls

In Chapter 5, you learn how to use the UpdatePanel control to quickly and easily add AJAX functionality to your ASP.NET web pages. You also learn how to use the Timer control to make asynchronous callbacks from your pages at a specified time interval.

Chapter 6—Advanced Techniques with the ASP.NET AJAX `PageRequestManager` Object

In Chapter 6, you learn some essential techniques that can be used with the ASP.NET AJAX `PageRequestManager` object to save you many lines of code as well as help you create richer user experiences for your AJAX-enabled applications.

Chapter 7—Using the ASP.NET AJAX Control Toolkit

In Chapter 7, you learn about the controls that are available in the ASP.NET AJAX Control Toolkit, which is an open-source, community-based suite of controls for ASP.NET AJAX.

Chapter 8—Building an ASP.NET AJAX Extender Control

Chapter 8 first introduces the concept of creating AJAX-enabled extender controls, and then shows you how to build an extender control that can be used in your applications.

Chapter 9—ASP.NET AJAX and SharePoint 2007

In Chapter 9, you learn how to enable ASP.NET AJAX in SharePoint 2007 and Windows SharePoint Services (WSS) 3.0, and then you learn how to build an AJAX-enabled WSS web part.

Chapter 10—Creating ASP.NET AJAX-Enabled Vista Sidebar Gadgets

In Chapter 10, you learn how to build ASP.NET AJAX-Enabled Gadgets for the Windows Vista Sidebar.

CHAPTER 1

Introduction to AJAX Technologies

IN THIS CHAPTER

If you've purchased this book, you probably are interested in AJAX technologies. If you are not familiar with the technology or are new to AJAX, it is important that you take some time and understand where AJAX fits into the big picture of web development. This section helps bring you up to speed with AJAX's place in your web development toolbox.

One problem with the design of web pages (especially ASP.NET web pages) is that to refresh the data on the page, it must postback to the server. This causes the page to flicker while the page is posting, and then the page is reloaded in the browser with the results of the post. You can view the amount of data with tools such as IEHTTPHeaders and HTTPWatch. You will quickly notice that the amount of information getting posted is quite sizeable because ASP.NET applications not only postback the controls, but also post the page's ViewState. Although the technique of a postback works, it creates a lot of traffic over the wire and inherently reduces the overall scalability of your application.

Asynchronous JavaScript and XML (AJAX) is a development pattern that you can use to provide your users with a much richer user experience in your web applications. Simply stated, AJAX allows you to asynchronously load data into pieces of a page on demand instead of loading the whole page each time data is requested.

An example of where a user experience can be enhanced by AJAX is a web page that contains related dropdown boxes. For example, say that you have a web page that supplies information on cars in which a user must select the year, make, and model of his car from dropdown boxes on the page. When a user selects a year, she populates all makes for a selected year in the make dropdown box. Additionally, when a user selects a make of car, the models dropdown is populated with models for a given year and make.

If you use traditional ASP.NET without AJAX, you will probably set the `AutoPostBack` property of your dropdowns to true and the page will postback (and flicker) when your user makes selections on the page.

Conversely, if you use AJAX, data can be loaded asynchronously into the list boxes as the user makes selections on the page. This is much more efficient because only the data being requested will travel in the request to the server, which could be as simple as a query string appended to the end of a page request. Also, the page will not flicker as the user makes selections in the dropdowns because the post actually is happening in the background.

NOTE

An example of AJAX is described later in the chapter.

AJAX and Web 2.0

Web 2.0 is a term (or rather buzzword) that you often hear when describing most "modern" web sites; however, it shouldn't be a new concept to web developers. Web 2.0 is actually a consolidation of many existing technologies that allows you to provide a rich interactive user experience over the web. Examples of Web 2.0 technologies include, but aren't limited to, the following areas:

- Rich Internet Applications (RIAs), which include AJAX, Adobe Flash, Silverlight, and Moonlight
- Web services
- Blogs
- Wikis
- Social networking
- Social bookmarking
- RSS/Atom

Before the Web 2.0 movement began on the Internet, web pages often focused solely on providing the user with data. The user would simply request a page, view the page, request another page, view that page, and so on.

In contrast, the patterns and techniques behind Web 2.0 are all about the user experience with the web: AJAX and web services for rich, efficient user experiences, blogs, wikis, social networking, and social bookmarking for collaboration, and RSS/Atom so that users can "subscribe" to data.

As technologies such as AJAX evolve and are adopted in large scale on the web, Web 2.0 techniques are quickly becoming the expected user experience for the web. Mainstream examples of AJAX include the Google-based applications, such as Google's Maps, Docs, and Calendar, as well as Microsoft-based applications, such as Hotmail, and Windows Live-based applications. As users start utilizing these types of applications in their everyday lives, they will come to expect the same type of functionality in the applications you develop.

Why Use AJAX?

As stated previously in this chapter, you can use AJAX to help provide a rich user experience. Of all the new, cool techniques and technologies that are available in Web 2.0 and rich Internet applications, AJAX is clearly the most widely used today. Should you use AJAX simply because it's cool? The short answer is "No," which is explained in the next section.

AJAX Rationale

Although the user experience that results from AJAX development patterns is a much richer experience, the rationale for utilizing AJAX is the total amount of traffic that is reduced from users accessing your web pages. When you have a web application with a large user base, using AJAX significantly increases the scalability of the application due to the amount of web traffic that is reduced every time a page is loaded.

The following list gives advantages and disadvantages of AJAX:

Advantages

- Reduced page bandwidth
- Load page data on demand
- Service-based approach to web development
- Rich user experience

Disadvantages

- Unexpected browser functionality (when the user clicks the back or refresh button)
- Business logic can exist in the JavaScript
- Difficult to test with automated tools

Although this book highlights and focuses on the advantages of AJAX, several disadvantages should be noted. The first thing your users will notice is that when they click the refresh or back button in the browser, the page has a possibility of losing its state when the page reloads. This is a result of dynamically loading data on demand (which was one of the advantages of AJAX).

The second disadvantage of AJAX that you will see in the development world is actually JavaScript. You might wonder that if AJAX is initiated from client-side JavaScript, how can JavaScript be a disadvantage? The answer is that JavaScript eventually will be considered a disadvantage for AJAX in your web applications. Over time, you will be quite shocked at the amount of JavaScript that is required (which is an evolution throughout the lifecycle of your application) for AJAX functionality in your application. It will start out very small and clean, but can quickly spiral out of control.

From the architecture perspective, there is not a clean way to reverse-engineer JavaScript into an architecture model. An architecture document will probably define how a page should function, and there could be thousands of lines of JavaScript behind the scenes to enable this functionality. Depending on your environment and Software Development Lifecycle (SDLC), this could become a problem because it is very difficult to validate the JavaScript against the architecture models.

The third (and certainly not the last) disadvantage is the potential difficulty of testing AJAX functionality with automated tools. For example, Visual Studio Team Suite (VSTS) 2005 had limited support for AJAX when creating web tests. This problem was easily circumvented with tools such as Fiddler (http://www.fiddlertool.com/fiddler/), which helps you capture AJAX requests, which then can be loaded into the VSTS Web Test. It is important to note that these issues have been resolved in VSTS 2008.

This section is designed not to scare you away from AJAX, but to create an awareness of things that can affect you and your AJAX development experience. Best practices and warnings are highlighted in upcoming chapters so that you can be aware of the pitfalls that can show up in your applications.

In the next section, you learn the basics of implementing AJAX in your applications.

AJAX: An Example

Now that you understand the advantages and disadvantages of AJAX, it is helpful to learn about the code behind AJAX requests. This section introduces an example of using AJAX and JavaScript Object Notation (JSON).

The `XMLHttpRequest` Object

The main object behind AJAX is the `XMLHttpRequest` object. This object exposes functionality that allows you to send and receive data (usually XML-based, but not required) from client-side JavaScript code to a server page.

`XMLHttpRequest` has six methods, listed in Table 1.1, and seven properties, listed in Table 1.2.

Deep knowledge of the capabilities of the `XMLHttpRequest` object's properties and methods is required learning for any AJAX developer. At its core, every library that encapsulates AJAX functionality (ASP.NET AJAX, AJAX.NET, Yahoo YUI, and so on) uses the `XMLHttpRequest` object.

The next section defines a simple AJAX JavaScript library that can be used to make AJAX calls in your web applications.

TABLE 1.1 `XMLHttpRequest` Methods

Method	Parameters	Description
`Abort`		Aborts the request that has been made.
`getAllResponseHeaders`		Returns a string of all response headers.
`getResponseHeader`	`headerName`	Returns the value of a given response header.

`Open`	`method`, `url`
	`method`, `url`, `async`
	`method`, `url`, `async`, `username`
	`method`, `url`, `async`, `username`, `password`
		Opens the request to a specified URL.
		`method` defines the HTTP method that will be used (typically GET or POST).
		`url` defines the location of the page that is being requested.
		`async` is a Boolean that defines whether the request will be made asynchronously or not.

`Send`	`content`
	`DOMString`
	`Document`
		Sends the request to the server.
		`DOMString` is the XML that is to be sent as a string.
		`Document` is an XML DOM document

`setRequestHeader`	`label, value`	Allows you to add a label and value to the HTTP header that is being sent to the server.

TABLE 1.2 XMLHttpRequest Properties

Property	Description
`onreadyStatechange`	Event handler that is fired when the `readyState` property changes. This will be a JavaScript function that handles the data that is returned to the client.
`readyState`	Returns the state of the `XMLHttpRequest` object. Possible values are 0 = not initialized 1 = open 2 = HTTP Headers received 3 = receiving 4 = loaded
`responseText`	Returns the complete response as a string.
`responseXML`	Returns the complete response (if it is XML) as an XML document object.
`responseBody`	Returns a binary encoded string of the response.
`Status`	Returns the HTTP status code of the request. HTTP status codes are defined at http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html.
`statusText`	Returns the HTTP status description of the request.

A Simple AJAX Library

As a developer, you probably have several (possibly hundreds of) scripts stuffed away somewhere that you reuse to accomplish specific tasks. One such script you can use for making AJAX calls is provided in Listings 1.1 through 1.3.

NOTE

Although this library can be written many different ways, this example contains three JavaScript functions: `createAjaxRequest`, `createHttpRequestObject`, and `getResponse`. Each function is separated into its own listing and described in the next three sections.

LISTING 1.1 `createAjaxRequest` Function

```
//Section 1
var http_request = false;

function createAjaxRequest(url, parameters) {
  http_request = createHttpRequestObject();

  if (!http_request) {
     alert('Error creating XMLHTTP object');
     return false;
  }
  http_request.onreadystatechange = getResponse;
  http_request.open('GET', url + parameters, true);
  http_request.send(null);
}
```

This section contains the `createAjaxRequest` function. This method is called when you want to invoke an AJAX request. When this method is invoked, it first makes a call to `createHttpRequestObject` (described in Listing 1.2) to create an instance of the `XMLHttpRequest` object. After the object is returned, you need to do some error checking around this object to ensure that it was created correctly. Next (and most importantly), the object is configured by setting the `onreadystatechange` property to the `getResponse` function handler (described in Listing 1.3), and the request is opened (as asynchronous) and finally sent to the server.

LISTING 1.2 `createHttpRequestObject` Function

```
//Section 2
function createHttpRequestObject(){
    var request;
    if (window.XMLHttpRequest) { // IE7, Mozilla, Safari, etc.
         request = new XMLHttpRequest();
         if (request.overrideMimeType) {
            request.overrideMimeType('text/xml');
         }
      } else if (window.ActiveXObject) { // IE 6 and previous
         try {
            request = new ActiveXObject("Msxml2.XMLHTTP");
         } catch (e) {
            try {
               request = new ActiveXObject("Microsoft.XMLHTTP");
            } catch (e) {}
         }
      }
      return request;
}
```

This section contains a helper function called `createHttpRequestObject`. Unfortunately, depending on the type of browser that you are using, there are many ways that you can (or should) create an instance of `XMLHttpRequest`. This method creates and returns the object instance of `XMLHttpRequest` that your browser supports.

LISTING 1.3 `getResponse` Function

```
//Section 3
function getResponse() {
  if (http_request.readyState == 4) {
     if (http_request.status == 200) {
        var xmldoc = http_request.responseXML;
        alert(xmldoc.xml);
     } else {
        alert('There was a problem with the request.');
     }
  }
}
```

Although `createAjaxRequest` and `createHttpRequestObject` are somewhat boilerplate functions, the processing of the response occurs in the `getResponse` function, defined in Listing 1.3, `getResponse`.

The handler to execute this function was set up in the Section 1 `createAjaxRequest` method by setting the `onreadystatechange` property of the `XMLHttpRequest` object. This function gets fired *every time* the `readyState` property changes. It important to note that you will not be able to access the response until the `readyState` property is equal to 4, or loaded, so you first must check the status of the `readyState` before processing the response.

After your response has been loaded, you must perform another check on the HTTP status. If anything is wrong with your request (for example, 401-unauthorized, 404-page not found, and so on), then you can respond to that error here. In the example in Listing 1.3, the status is checked for a value of 200 (or OK), and then the function will process the request.

As soon as you make these two checks on the `readyState` and `status` properties of the `XMLHttpRequest` object, you can then process the response. In this example, the response is loaded into an XML Document object, and then the XML is displayed in a message box to help confirm the results.

Listing 1.4 is a complete code listing for all of the code described in this section.

LISTING 1.4 Simple AJAX Library–Complete Listing

```
var http_request = false;


function createAjaxRequest(url, parameters, callbackFunction) {
  http_request = createHttpRequestObject();
  if (!http_request) {
     alert('Error creating XMLHTTP object');
     return false;
  }
  http_request.onreadystatechange = getResponse;
  http_request.open('GET', url + parameters, true);
  http_request.send(null);
}

function createHttpRequestObject(){
    var request;
    if (window.XMLHttpRequest) { // IE7, Mozilla, Safari, etc.
         request = new XMLHttpRequest();
         if (request.overrideMimeType) {
            request.overrideMimeType('text/xml');
         }
      } else if (window.ActiveXObject) { // IE 6 and previous
         try {
            request = new ActiveXObject("Msxml2.XMLHTTP");
         } catch (e) {
            try {
               request = new ActiveXObject("Microsoft.XMLHTTP");
            } catch (e) {}
         }
      }
      return request;
}

function getResponse() {
  if (http_request.readyState == 4) {
     if (http_request.status == 200) {

        var xmldoc = http_request.responseXML;
        alert(xmldoc.xml);
     } else {
        alert('There was a problem with the request.');
     }
  }
}
```

NOTE

You will most likely process the XML document and not display it in an alert. An example of processing the returned XML is explained in the next section.

Using the AJAX Library

After you understand the basics of AJAX, you can start applying this knowledge to your projects. In this section, you learn how to use AJAX to connect two dropdown list boxes in ASP.NET.

This example allows you to select a make of car (BMW, Mercedes, or Porsche) and use AJAX to request the models for the selected car make. The very simple page is illustrated in Figure 1.1.

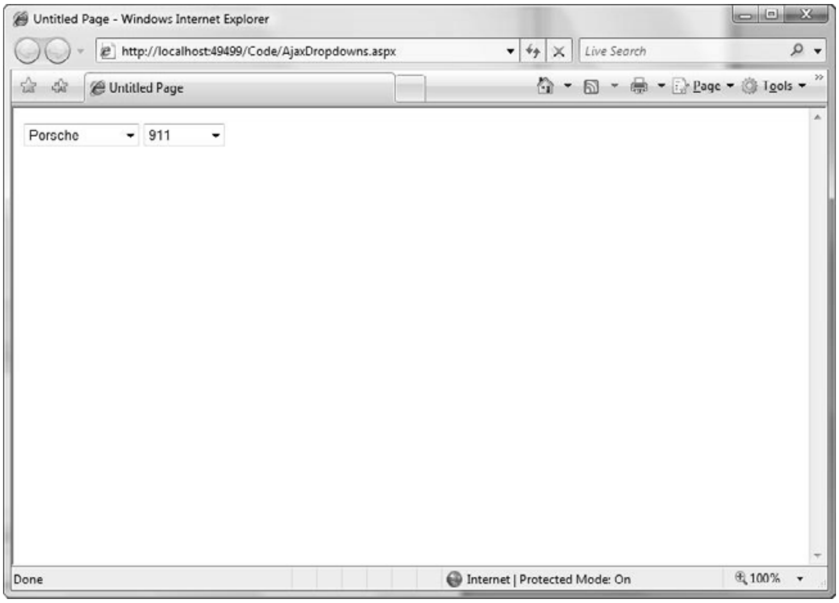
Untitled Page - Windows Internet Explorer
http://localhost:49499/Code/AjaxDropdowns.aspx
Live Search
Untitled Page
Page
Tools
Porsche
911
Done
Internet | Protected Mode: On
100%

FIGURE 1.1 AJAX dropdowns

When you select a car from the first dropdown, an AJAX request is made to the server to get different car models. For this example, a call is be made to a page called getModels.aspx, which is defined in Listing 1.4.

LISTING 1.4 getModels.aspx

```
<%@ Page Language="C#"%>


<script runat="server">


    protected void Page_Load(object sender, EventArgs e)
    {
        Response.ContentType = "text/xml";
        string Make = Request.QueryString["make"];
        switch (Make.ToUpper())
        {
            case "BMW":
                Response.Write("<models><car>3-Series</car>
<car>5-Series</car><car>7-Series</car><car>M</car><car>X3</car>
<car>X5</car><car>Z-Series</car></models>");

                break;

            case "MERCEDES":
                Response.Write("<models><car>C-Class</car>
<car>E-Class</car><car>S-Class</car><car>AMG</car></models>");

                break;

            case "PORSCHE":
                Response.Write("<models><car>911</car><car>Cayman</car>
<car>Cayanne</car><car>Boxster</car></models>");

                break;

            case "BLANK":
                Response.Write("<models><car></car></models>");
                break;

            default:
                Response.Write("<models><car>Invalid selection</car></models>");

                break;

        }

    }
</script>
```

This page simply accepts a query string argument called `make` and will return an XML string of car models, depending on the `make` argument. A sample of the returned XML where the value `PORSCHE` is passed into the `make` query string argument is described in Listing 1.5.

LISTING 1.5 Models XML Records

```
<models>
    <car>911</car>
    <car>Cayman</car>
    <car>Cayanne</car>
    <car>Boxter</car>
</models>
```

NOTE

Note that this syntax is important, because you will learn how to parse the XML document and populate the second dropdown with the values that are returned from this page.

Now that you have learned where the data is coming from, it is time to learn how to request and manipulate the data using AJAX. Listing 1.6 describes a page named AjaxDropdowns.aspx.

LISTING 1.6 AjaxDropdowns.aspx

```
<%@ Page Language="C#"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

    protected void Page_Load(object sender, EventArgs e)
    {
        //setup the AJAX call on the dropdown
        ddlMakes.Attributes.Add("onchange",
"javascript:createAjaxRequest('getModels.aspx', '?make=' + " + ddlMakes.ClientID +
".value);");

    }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Untitled Page</title>

</head>
<body>
    <form id="form1" runat="server">
```

```
<div>

    <asp:DropDownList ID="ddlMakes" runat="server">
        <asp:ListItem Value="blank">Select a Make</asp:ListItem>
        <asp:ListItem>BMW</asp:ListItem>
        <asp:ListItem>Mercedes</asp:ListItem>
        <asp:ListItem>Porsche</asp:ListItem>
    </asp:DropDownList>
    <asp:DropDownList ID="ddlModels" runat="server">
    </asp:DropDownList>
</div>
</form>
<script type="text/javascript">
//Section 1
var http_request = false;

function createAjaxRequest(url, parameters) {
  http_request = createHttpRequestObject();

  if (!http_request) {
     alert('Error creating XMLHTTP object');
     return false;
  }
  http_request.onreadystatechange = getResponse;
  http_request.open('GET', url + parameters, true);
  http_request.send(null);
}

//Section 2
function createHttpRequestObject(){
    var request;
    if (window.XMLHttpRequest) { // IE7, Mozilla, Safari, etc.
         request = new XMLHttpRequest();
         if (request.overrideMimeType) {
            request.overrideMimeType('text/xml');
         }
      } else if (window.ActiveXObject) { // IE 6 and previous
         try {
            request = new ActiveXObject("Msxml2.XMLHTTP");
         } catch (e) {
            try {
               request = new ActiveXObject("Microsoft.XMLHTTP");
            } catch (e) {}
         }
      }
```

LISTING 1.6 Continued

```
            return request;
    }

    //Section 3
    function getResponse() {
      if (http_request.readyState == 4) {
         if (http_request.status == 200) {
            //get dropdown
            var ddl = document.forms[0].<%=this.ddlModels.ClientID %>;

            //clear items in dropdown
            ddl.options.length = 0;

            //get the XML response and loop elements
            var xmldoc = http_request.responseXML;
            var cars = xmldoc.getElementsByTagName('car');
            for(i=0; i<cars.length; i++){
                //add items to the dropdown
                try{
                    var opt = new
Option(cars[i].firstChild.nodeValue, cars[i].firstChild.nodeValue);
                    ddl.add(opt, i);
                }
                catch (e){
                    ddl.length=0;
                }
            }
            http_request = null;
         }
         else {
            alert('There was a problem with the request.');
         }
      }
    }

</script>
</body>
</html>
```

As you browse through the code in Listing 1.6, first notice that it is using the simple AJAX library that you learned about earlier in this chapter. You learn about the changes that were made to the `getResponse` function later in this chapter, but it is important to note because you will also see calls made to the `createAjaxRequest` function, which is discussed next.

This page contains two dropdowns, ddlMakes and ddlModels, and one server side event handler, Page_Load. When the user selects a car make from the ddlMakes dropdown, an AJAX call needs to be made to populate the ddlModels dropdown with car models. In this example, the AJAX call is set up on the ddlMakes dropdown in the Page_Load event by adding an onchange attribute to the server side dropdown list control and setting its value to call the createAjaxRequest JavaScript method defined in the simple AJAX library.

The big change to the AJAX library is in the getResponse method, which is highlighted in Listing 1.7.

LISTING 1.7 getResponse Method

```
function getResponse() {
      if (http_request.readyState == 4) {
         if (http_request.status == 200) {
            //get dropdown
            var ddl = document.forms[0].<%=this.ddlModels.ClientID %>;

            //clear items in dropdown
            ddl.options.length = 0;

            //get the XML response and loop elements
            var xmldoc = http_request.responseXML;
            var cars = xmldoc.getElementsByTagName('car');
            for(i=0; i<cars.length; i++){
                //add items to the dropdown
                try{
                    var opt = new
Option(cars[i].firstChild.nodeValue, cars[i].firstChild.nodeValue);
                    ddl.add(opt, i);
                }
                catch (e){
                    ddl.length=0;
                }
            }
            http_request = null;
         }
         else {
            alert('There was a problem with the request.');
         }
      }
    }
```

As you walk through the method, first, a reference is made to the ddlModels dropdown on the page, and all items are cleared by setting the options.length property to zero. If you don't do this, items will simply get added to the dropdown list every time the user selects a new car make. Next, the XML response is returned from the `http_request` object. Finally, the code loops through the values of all the returned `<car>` elements and populates the `ddlModels` dropdown list.

When you execute the page, you will notice that the dropdown is populated with data very quickly with no page flicker. This is a great first example of using AJAX because linked (or related) dropdowns are a common scenario that you see in applications. Additionally, functionality such as this is often a developer's first experience with AJAX functionality in their applications.

Before ASP.NET AJAX (and other libraries), this was how we, as a development community, had to write AJAX functionality. One of the first things you will notice about this approach is the amount of JavaScript that you have to write and maintain. It quickly becomes very difficult to maintain and add or change the functionality of your pages without starting from scratch. There is nothing wrong with this approach, and libraries such as ASP.NET AJAX shield you from a lot of the coding and JavaScript maintenance (and the libraries *are* generating this code for you). However, it is important to note the maintenance cost associated with manually writing AJAX code.

Summary

In this chapter, you learned about AJAX and its place in the Web 2.0 world. AJAX has a place in your web applications, and the rationale of utilizing AJAX was discussed in detail as well as the advantages and disadvantages that you will encounter by applying this type of functionality in your applications. Finally, you learned how to build AJAX from scratch using ASP.NET and JavaScript.

Index