7 Developing ASP.NET AJAX Applications

In Depth

You are already familiar with the Web applications that use the `postback` model or method to refresh the content of a page. In the `postback` model, with every request made by an application viewer, the entire page is sent back to a 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.

The `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 by providing the ability to perform complex tasks and update the Web pages quickly. However, as compared to Windows application, Web applications are still less interactive and responsive. 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 they can update the page information faster, without page flicker and without interrupting user interactions.

In the real world scenario, you can take an example of Google E-mail Service, i.e. Gmail. In Gmail, when you need to attach a document with the text mail, you 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, you need not wait to write the text message until the document is completely attached with the mail. The processing of attaching the document in background is possible with a new technology called Asynchronous JavaScript and XML (`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 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 modifying the rest of the page.

In this chapter, you learn about AJAX and how it is different from other technologies. Next, you learn about ASP.NET AJAX, its components, and ASP.NET technology that use the `AJAX` approach for Web development. You also learn about the extension controls and enhancements in AJAX delivered with Visual Studio 2010, and their use to implement `AJAX` functionality in your Web application.

Exploring AJAX

AJAX is neither a new programming language nor a new platform for developing websites. You can call `AJAX` a new technique because it displays the refreshed content on a Web page by using the Page Update approach rather than the Page Replacement approach, which was used in traditional websites. `AJAX` is built on existing technologies, such as Document Object Model (DOM), Cascading Style Sheet (CSS), and Extensible Hypertext Markup Language (XHTML). It is a collection of development components, tools, and techniques used to create highly interactive Web applications. AJAX uses client scripts to make asynchronous calls to a server and load only those parts of a Web page on the client machine that needs to be changed. The transfer of data from the server is carried out over the Extensible Markup Language/ Hypertext Markup Language (`XMLHTTP)` protocol in the form of packets by using Extensible Markup Language (XML) and JavaScript Object Notation (JSON). AJAX is built using various technologies, which are as follows:

- JavaScript
- XHTML
- CSS
- DOM
- XMLHttpRequest object

Let's discus these technologies in detail, one by one.

Introducing JavaScript

`JavaScript` is a client-side dynamic scripting language that supports object-oriented programming. It provides various tools, such as form elements, to communicate with a server. There are 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 Hypertext Markup language (HTML) pages. It is an interpreted language, and not a compiled language. It does not require static type checking as in C++ and C#. In JavaScript, you can declare a variable without specifying its type. This makes it easier to work with JavaScript.

Introducing XHTML

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

XHTML is used with AJAX-enabled website to ensure that the website runs consistently on all browsers, including desktop browsers (IE, Mozilla Firefox, and Opera) and browsers for the hand-held devices Wireless Application Protocol (WAP) browser. XHTML is preferred over HTML because sometimes you come across Web pages containing inappropriate HTML tags, such as tags that do not have closing tags. 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. Using CSS, you can specify fonts, colors, styles (bold, italic), and size of controls. 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 file with the Web Form. The following code snippet shows how to define styles using 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. AJAX includes CSS as it provides the following advantages:

- Allows you to 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.
- Separates the working of form coding and designing, which in turn makes it easy for other developers to understand both the definition and design system of a Web Form clearly.
- Helps in the processing of further requests on the same website quickly as CSS is cached in the browser memory in the first visit to the website.
- 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. It is a platform and language-independent interface that allows programs and scripts to access and update the content of a Web Form. DOM is recommended by World Wide Web Consortium (W3C) to manipulate content at runtime. It 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 and child nodes. This can be understood with the help of Figure 7.1:

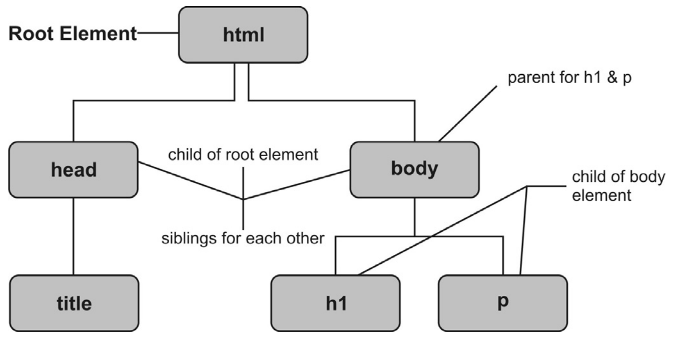


Figure 7.1: Displaying the Tree Representation of HTML Document

Figure 7.1 shows various nodes of a Web Form and the relationship between the nodes.

`DOM` has a rich set of methods, such as `insertBefore`, `insertAfter`, and `hasChildNodes`, which 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 content of a Web page, whenever required.

Introducing the XMLHttpRequest Object

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

Noteworthy properties of the `XMLHttpRequest` object are listed in Table 7.1:

Table 7.1: Noteworthy Properties of the XMLHttpRequest Object

Property	Description
`readyState`	Retrieves the current state of the request operation
`responseBody`	Retrieves the response body as an array of unsigned bytes
`responseText`	Retrieves the response body as a string
`responseXML`	Retrieves the response body as an XML DOM object
`Status`	Retrieves the HTTP status code of a request
`statusText`	Retrieves the HTTP status of a request

Noteworthy methods of the `XMLHttpRequest` object are listed in Table 7.2:

Table 7.2: 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 a method, destination URL, and other optional attributes of a pending request to the XMLHttpRequest object
`send()`	Conveys an HTTP request to the server and retrieves a response
`setRequestHeader()`	Inserts custom HTTP headers in a request

NOTE

The 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 to receive response from a server. IE 8 provides an event named ontimeout to the XMLHttpRequest object to instruct a browser what to do when time-out period expires.

Let's now understand the need for AJAX.

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 tedious request-response model. `AJAX` prevents the developers from designing more Web pages for displaying new information to the user. This is possible because `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. Using AJAX is money-saving too, as users 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, and Firefox 1.0 and advanced versions. AJAX-enabled websites can work easily even with slow connections, such as dial-up connection.

ASP.NET AJAX fulfills its aim of providing a comprehensive Web development environment that is rich in client-scripting features and supports ASP.NET server-side development. ASP.NET 4.0 has built-in 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 and 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, you first need to analyze how other technologies work. Let's understand the working of traditional Web applications with the help of Figure 7.2:

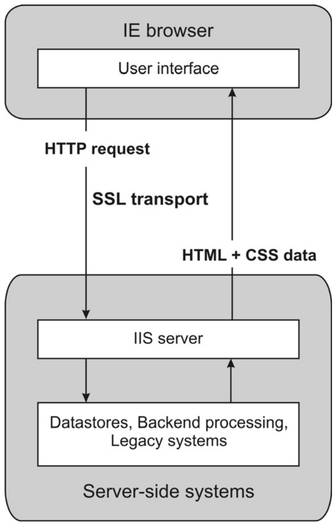


Figure 7.2: Displaying the Traditional Web Application Model

As shown in Figure 7.2, 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 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 7.3 shows the `AJAX` Web application model:

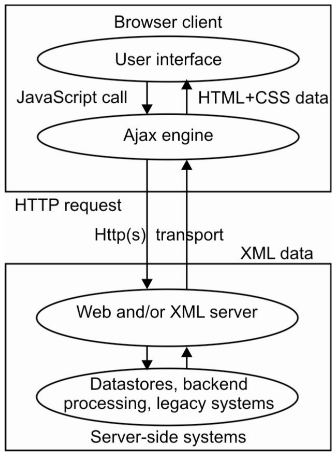


Figure 7.3: Displaying the AJAX Web Application Model

The AJAX Web application model handles the client-side activities, such as validations, without making a round trip to the server. The tasks, which are required to be processed at the server-end, are handled by the AJAX engine at the client-side. 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.

Exploring New Features of AJAX

The AJAX framework has given foundation for building efficient websites after its inclusion in the ASP.NET framework. Therefore, Microsoft keeps enhancing the features of AJAX with every release of ASP.NET to provide more functionality to build rich Internet applications. ASP.NET 4.0 AJAX includes new features that provide more functionality to a user. These features are as follows:

- Support for live data binding
- Support for client-side template rendering
- Support for declarative instantiation of client components
- Support for using the observer pattern on JavaScript objects and arrays
- Support for invoking ADO.NET data services and data contexts
- Support for the DataView control

Let's discuss each of these features in detail.

Support for Live Data Binding

Live data binding is feature which ensures that the data is bound with the server controls at runtime. ASP.NET AJAX 4.0 provides the support for live data binding with the help of observer pattern. The Sys.Observer interface is used to support live data binding. The live data binding can be used in different ways, one way and two way. In one way data binding, the data is not updated automatically if there is any change in the data source. On the other hand, in the two way data binding, the data is updated automatically if the data source changes.

The code snippet for one way data binding is as follows:

```
<h3>{{ ProducttName }}</h3>
```

The code snippet for two way data binding is as follows:

```
    <input type="text" value="{binding ProductName}"/>
```

Support for Client-Side Template Rendering

In the client-based development environment, templates are the efficient way of developing user interface (UI). A new template engine is included in ASP.NET 4.0 used to execute applications at the client-side. The template engine includes the following features:

- **Performance—**States that the engine must be powerful enough to render a number of items using an advance template before interruption occurs while communicating with the application.
- **Simplicity—**States that the template syntax must be easy and used for the most common scenario, such as for one-way/one-time binding.
- **Expression language—**States that the template must support an expression language to use simple syntax, preferably JavaScript syntax.
- **XHTML compliance—**States that the template should render XHTML-standard markup.
- **Components—**States while using the template syntax, the developer can trigger client-side controls and behaviors that are bound to HTML elements.

In ASP.NET AJAX 4.0, an HTML template is a DIV tag, or any other container tag, declared with the sys-template class attribute, as shown in the following code snippet:

```
    <div>
    <ul id="MyTemplate" class="sys-template">
           <li>
             {{ Symbol }}, {{ Quote }}, {{ Change }}
           </li>
    </ul>
    </div>
```

The sys-template class attribute initially marks an element and its content invisible. The sys-template must be defined explicitly in the master page or in every page, as shown in the following code snippet:

```
    <style type="text/css">
    .sys-template { display:none; visibility:hidden; }
    </style>
```

Support for Declarative Instantiation of Client Components

ASP.NET AJAX 4.0 supports declarative instantiation of client components, such as controls and behavior attached to them.

The declarative markup includes additional namespace attributes which follow XHTML compliant. The declarative instantiation of controls works only when the declarative markup is present within an element that has been configured for this purpose only.To declaratively instantiate controls outside a template, you need to first configure a page to ensure that the sys:attach markup is present within an activated element. The following code snippet shows how to activate controls:

```
    <body xmlns:sys="javascript:Sys" sys:activate="Button1,Button2">
```

In the preceding code snippet, we have activated Button controls.

You can also activate every element in the document but it causes a delay in initialization of the page, therefore, this technique should not be used when pages contain a large amount of data. The following code snippet shows how to activate every element of a document:

```
    <body xmlns:sys="javascript:Sys" sys:activate="*">
```

Support for the Observer Pattern on JavaScript Objects and Arrays

The observer pattern is used to implement functionality in which any modifications to JavaScript objects raise notifications that can be handled in your code. The observer design pattern helps to set observers on an object. The state change of objects is handled by the respective event handlers. It can also be used to establish live bindings between UI elements and objects or arrays, which might be provided by the JSON Web service.

ASP.NET AJAX 4.0 uses the Sys.Observer class to support the observer design pattern. The following code snippet shows how to use the Sys.Observer class:

```
    var product = {pname: "Icecream", ID: "SU",order:true }
    var callproduct= Sys.Observer.observe(product);
    Sys.Observer.addPropertyChanged(callproduct, productSubscriber);
    employeePublisher.setValue("order",false);
    function employeeSubscriber(product, eventArgs)
    {
        if (eventArgs.get_propertyName() == "order" && product.order == false)
        {
            alert("This product is not available in the stock");
        }
    }
```

Support for Invoking ADO.NET Data Services and Data Contexts ASP.NET 4.0

The data context object or data context enables you to bind the JavaScript array or object to a template. It can interact with JSON-based Web Services. There are two types of data contexts provided by the ASP.NET AJAX, Sys.Data.DataContext and Sys.Data.AdoNetDataContext.

The data context recognizes all the changes in the data automatically by using new Sys.Observer object. The AdoNetDataContext data context supports enhanced features of ADO.NET data services, such as identity management, links, and association between entity sets. The following code snippet shows how to interact with Productlistservice ADO.NET data service:

```
    var dataContext = new Sys.Data.AdoNetDataContext();
    dataContext.set_productserviceUri("Productlistservice.svc");
    dataContext.initialize();
```

In the preceding code snippet, the set_poductserviceUri() method is used to interact with WCF AJAX or ADO.NET data service. The initialize() method is used to initialize the data context.

Support for the DataView Control

The DataView control is used to access data. It is bound with live data, which can be defined as data at runtime. Moreover, you can use this control to develop a dynamic data driven user interface. This control is defined in the System.UI.DataView namespace and contains two important properties, data and dataprovider. The data property is used to bind a JavaScript object with the data and the dataprovider property is used to connect to a WCF service. The following code snippet shows how to use the DataView control in ASP.NET AJAX 4.0:

```
    <body xmlns:sys="javascript:Sys"
        xmlns:dataview="javascript:Sys.UI.DataView"
        sys:activate="*">
        <ul sys:attach="dataview" class="sys-template"
            dataview:data="{{ dataArray }}">
            <li>            <h3>{{ProductName }}</h3>
                <div>{{ Price }}</div>
         </li>
           </ul>
    </body>
```

Let's discuss the features that are included in the Microsoft AJAX Library.

New Features in the Microsoft AJAX Library

The Microsoft AJAX library is a client-based JavaScript library that is compatible with all modern browsers and offers a lot of functionality as compared to JavaScript. You can use the Microsoft AJAX library to build highly responsive and interactive database-driven Web applications that can run entirely within the Web browser. The Microsoft AJAX library can be used with both ASP.NET Web Forms and ASP.NET MVC applications. This library is released with new features and fully supports ASP.NET 4.0. The new features included in the Microsoft AJAX library are as follows:

- Imperative syntax
- Clientscript loader

- Client data access
- Client datacontext and AdoNetDataContext classes
- jquery integration

Let's discuss these features in detail, one by one.

Imperative Syntax

The Microsoft AJAX library in ASP.NET 4.0 now supports simple imperative syntax that is used to create and manage controls. The following code snippet shows how to create and attach a watermark to an HTML element whose ID is ProductName:

```
    Sys.create.watermark("#ProductName", {WatermarkText: "Add productname here..." } );
```

When an HTML document that contains the preceding code snippet is executed in a Web browser, a watermark is displayed.

Clientscript Loader

The Microsoft AJAX library in ASP.NET 4.0 includes a script loader, also known as Clientscript loader. The Clientscript loader retrieves all scripts that are needed by one or more client component or control automatically and executes the scripts in the order in which they are received. The client-script loader provides the following advantages:

- Loads all the resources that are needed by a script automatically
- Ensures that each script is loaded only once
- Enhances the performance of an AJAX application by loading scripts in parallel and by combining scripts
- Loads the scripts only when they are needed, referred as lazy loading
- Loads third-party scripts, such as jQuery and user scripts

Client Data Access

The Microsoft AJAX library in ASP.NET 4.0 supports building of database-driven Web applications that can execute in a Web browser as well as allows you to access client data. The three features of Microsoft AJAX that are used to access client data are as follows:

- **Client data control**—Refers to a control that is used to display either a single database record or a set of records. You can display database records using the DataView control, which is a type of client data control.
- **Client template**—Helps you display database records by using the DataView control.

Client DataContext and AdoNetDataContext Classes

The Microsoft AJAX library in ASP.NET 4.0 includes a client DataContext class that is related to the server-side LINQ to SQL DataContext or the Entity Framework ObjectContext classes. The DataContext class provides the following advantages:

- Supports read and write permission to data from a database
- Supports identity management and change tracking
- Supports complex links and associations between entities from multiple entity sets or tables

The Microsoft AJAX library now includes a generic DataContext class used to interact with ASP.NET or WCF Web services. The Microsoft AJAX library also includes the AdoNetDataContext class that is designed to enable you to easily interact with an ADO.NET Data Services service. You can use either the DataContext or the AdoNetDataContext class to retrieve and update data in a database.

jQuery Integration

ASP.NET 4.0 AJAX supports jQuery integration. The jQuery library is an open-source JavaScript library that can be used with both ASP.NET Web Forms and ASP.NET MVC. The Microsoft AJAX library was designed to help you access the elements in your Web pages, work with client-side events, enable visual effects, and make it easier to use AJAX in your applications. jQuery provides flexible plug-in using which you need not to code the same functionality in your applications again and again. You can use both jQuery plug-ins and Microsoft AJAX client controls within the same AJAX application. You can use jQuery in your Web page, as shown by the following code snippet:

```
<script src="Scripts/jquery-1.3.2.js" type="text/javascript"></script>
<script type="text/javascript">
  $("input").focus(function () { $(this).css("background-color", "green"); });
  $("input").ready(function () { alert("Welcome to the world of jquery"); });
  $('#Button1').click(function () { $(this).css('background-color', 'red').animate({
  width: '100px', height: '80px' }) });
  $('h2').css('font-size', '40px').fadeOut(5000);
</script>
```

Let's now learn about ASP.NET AJAX architecture.

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. The ASP.NET AJAX architecture comprises the client-side architecture and server-side architecture. The ASP.NET AJAX client-side architecture is the client script that a Web application uses to call an application or a service on a server. The ASP.NET AJAX server-side architecture comprises ASP.NET Web services, ASP.NET server controls, and ASP.NET AJAX Extension controls. 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 a server asynchronously. This asynchronous call support is quite new in a scripting environment and introduces a high level of modularity in client scripting. The 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, the client-side architecture shares data with Web servers mostly in the form of JSON and XML format, because both these formats are platform-independent. Figure 7.4 shows the client-side architecture:

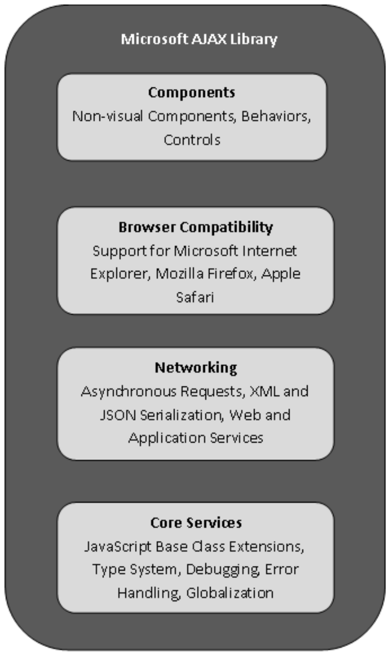

Microsoft AJAX Library
Components
Non-visual Components, Behaviors, Controls
Browser Compatibility
Support for Microsoft Internet Explorer, Mozilla Firefox, Apple Safari
Networking
Asynchronous Requests, XML and JSON Serialization, Web and Application Services
Core Services
JavaScript Base Class Extensions, Type System, Debugging, Error Handling, Globalization

Figure 7.4: Displaying the Client-side Architecture

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

- ❑ **Components**—Allows you to get robust functionalities without performing `postbacks`. The use of components varies as per the type of client behavior. There are the following three categories of components:
  - **Components**—Encapsulates the code to perform a task, but the task is not visible in a browser
  - **Behaviors**—Enhances the basic behavior of the existing `DOM` objects
  - **Controls**—Creates new `DOM` objects having custom behaviors
- ❑ **Browser Compatibility**—Specifies that AJAX applications are compatible with almost all browsers, such as 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**—Deals with the communication between script in the browser, 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**—`Supports` 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, and inheritance
  - A base class library that contains components, such as string builders
  - JavaScript libraries that are either embedded in an assembly or provided on the hard-disk drives
  - Globalization that enables the usage of a single code base to provide user interface for multiple locales

The core services also include the `Sys.Debug` class and error objects to support debugging and error handling. The `Sys.Debug` class provides methods to display objects in the readable form at the end of a Web page and show trace messages.

Server-Side Architecture

The ASP.NET AJAX server-side architecture contains a set of server controls and components to organize user interface and maintain flow of application's execution. It also contains Web servicesto authenticate and define roles as well as manage user profiles. It also manages serialization, validation, and control extensibility.

Figure 7.5 shows the server-side architecture:

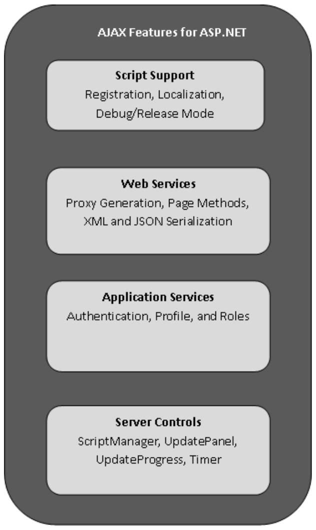

AJAX Features for ASP.NET
Script Support
Registration, Localization, Debug/Release Mode
Web Services
Proxy Generation, Page Methods, XML and JSON Serialization
Application Services
Authentication, Profile, and Roles
Server Controls
ScriptManager, UpdatePanel, UpdateProgress, Timer

Figure 7.5: Displaying the Server-side Architecture

The various components of the server-side architecture are as follows:

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

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

Script Support

The server-side architecture of ASP.NET contains a rich set of `JavaScript` files that are used to implement `AJAX` functionality in ASP.NET applications. These files are sent by Web server to client browser. However, scripts forwarded by the server vary according to the functionality implemented in a Web application and support partial-page rendering. The Microsoft AJAX library, in form of script, provides namespaces, inheritance, interfaces, and other related features to create a website. Sometimes while implementing `AJAX` functionality in the website, existing script files might be insufficient to accomplish a specific task. In such cases, you can create your own custom scripts. Such custom scripts can be saved as static `JavaScript` files or can be included as resources in an assembly. You need to register the custom scripts with Microsoft AJAX library to use them in your application. AJAX also supports localization; thereby, enabling you to create culture and language-specific applications. This localization support enables you to create your own script files for various languages and save them at a remote location. You can then selectively access these script files for a specific language and culture.

Web Services

A website that has `AJAX` functionality might need to call a Web service to implement certain functionality. For this purpose, you need to add the reference of certain script files from Microsoft Library in the Web page in which you need to call the Web service. Using the Visual studio IDE and AJAX server controls, the required script reference is automatically added to a 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 in a Web page. After you have added the reference of script files, the required proxy classes are automatically created. These classes call the Web service from the client-side. The Web services are also capable of serializing objects so that the objects can travel over the network. This serialization occurs in the form of XML or JSON because XML and JSON are standards for cross-platform and cross-browser data transfer.

Application Services

The tasks of authenticating users, authorizing them specific roles, and preparing their profiles are 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 a server
- `Membership`—Helps in authentication
- `Roles`—Implements role-based authorization for ASP.NET AJAX applications
- `Personalization`—Persists personalization data on a 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, such as a Button control, 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 server control ensures that client script reference is added to the Web page automatically. The client script reference helps to execute server controls.

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 4.0 and Visual Studio 2010. 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 a Web Form, the reference of corresponding script library is added automatically to the Web page. AJAX provides the following server controls:

- 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 helps in implementing the AJAX functionality in an ASP.NET website. You need to add this control on a Web page 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. This control uses a ScriptManager class to manage AJAX script libraries and script files. The inheritance hierarchy of the ScriptManager class is shown 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 a Web page by dragging the control from the Toolbox and dropping it on the Design view. The ScriptManager control is initialized using the following mark up code snippet:

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

Noteworthy properties of the ScriptManager class are listed in Table 7.3:

Table 7.3: 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

Table 7.3: Noteworthy Properties of ScriptManager Class

Property	Description
`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- page rendering
`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 representing a script file rendered to a client
`Services`	Retrieves a `ServiceReferenceCollection` object that contains a `ServiceManager` object for each Web service that ASP.NET exposes on a client for `AJAX` functionality
`SupportsPartialRendering`	Specifies a value to indicate whether or not a 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

Noteworthy methods of the `ScriptManager` class are listed in Table 7.4:

Table 7.4: Noteworthy Methods of ScriptManager Class

Method	Description
`GetCurrent()`	Obtains the `ScriptManager` instance for a given Web page object.
`GetRegisteredArrayDeclarations()`	Obtains a read-only collection of `ECMAScript` (JavaScript) array declarations that were previously registered with the Web page object.
`GetRegisteredClientScriptBlocks()`	Obtains a read-only collection of client script blocks that were previously registered with the `ScriptManager` control.
`GetRegisteredDisposeScripts()`	Obtains a read-only collection of dispose scripts that were previously registered with the Web page object.
`GetRegisteredExpandoAttributes()`	Obtains a read-only collection of custom attributes that were previously registered with the Web page object.

Table 7.4: Noteworthy Methods of ScriptManager Class

Method	Description
`GetRegisteredHiddenFields()`	Obtains a read-only collection of hidden fields that were previously registered with the Web page object.
`GetRegisteredOnSubmitStatements()`	Obtains a read-only collection of `onsubmit` statement that was previously registered with the Web page object.
`GetRegisteredStartupScripts()`	Obtains 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 a 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 a 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 a Web page. It is an overloaded method.
`RegisterClientScriptResource()`	Registers a client script that is embedded in an assembly with the `ScriptManager` control. It is an overloaded method.
`RegisterDataItem()`	Sends custom data to a control during partial-page rendering. It is an overloaded method.
`RegisterDispose()`	Registers 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 with the `ScriptManager` control. It is an overloaded method.
`RegisterOnSubmitStatement()`	Registers `ECMAScript` (JavaScript) code that is executed on submitting a 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.
`RegisterScriptDescriptors()`	Instructs the `ScriptManager` control to call back 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 a 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 7.5:

Table 7.5: Noteworthy Events of ScriptManager Class

Event	Description
`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

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 added in the Master page, then all the content pages (Web pages inherited from the Master page) use the ScriptManager control of the Master Page to manage its AJAX server controls. For this purpose, 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 with the Master page. 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 a 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 by using 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 as follows:

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

You can add the ScriptManagerProxy control in a Web page by dragging the control from the Toolbox and dropping it on the Design view. 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 content of a 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, i.e. 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. 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 content of more than one parts of a Web page. The Timer control's another useful property is the Enabled property. Using the Enabled property, 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, the Timer control can fire the Tick event; otherwise, not. The Timer control is an 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 a Web page by dragging the control from the Toolbox and dropping it on the Design view. 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 7.6:

Table 7.6: 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

The UpdatePanel Control

The UpdatePanel control helps to divide your Web page into parts, where each part can be updated independently. This control uses a synchronous postback to refresh the selected part of the page; but built-in client script transforms this synchronous request to an asynchronous request with the help of the XMLHttpRequest object, so that it can update the information without refreshing the page. 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 as follows:

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

You can add the UpdatePanel control in a Web page by dragging the control from the Toolbox and dropping it on the Design view. 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 7.7:

Table 7.7: Noteworthy Properties of UpdatePanel Class

Property	Description
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 control when a client triggers the asynchronous postback
RenderMode	Specifies a value to indicate whether or not the content of the UpdatePanel control is placed within the HTML <div> or <span> 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. The `UpdatePanel` control with a code sample is discussed in the Immediate Solutions section of this chapter.

The UpdateProgress Control

While working with the `UpdatePanel` control, sometimes you need to show the status of the information requested by a user. In other words, you need to make the user 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 of the requested data. 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. It is also useful when speed of updating content is slow. In this condition, the user gets idea how much information is processed and for how long, 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 a Web page by dragging the control from the Toolbox and dropping it on the Design view. 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 7.8:

Table 7.8: Important Properties of UpdateProgress Class

Property	Description
`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

To work with the UpdatePanel, UpdateProgress, and Timer Controls, you need to add the ScriptManager control on a Web page, because the ScriptManager control contains a reference of the built-in script libraries to implement AJAX functionality.

Let's now move on to the Immediate Solutions section to learn the practical implementation of various concepts discussed in this chapter.

Immediate Solutions

Differentiating Between AJAX and Non-AJAX Applications

ASP.NET 4.0 provides AJAX-specific controls to create AJAX-enabled website. Along with these controls, you can use other ASP.NET controls, such as Standard controls, Login controls, and Navigation controls. 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.

NOTE

Open ASP.NET Empty Website and add a Default.aspx page to it before creating any application.

Creating a Simple Non-AJAX Application

Let's perform the following steps to create a simple ASP.NET website that displays current system time on a button click:

1. Create an ASP.NET website and save it as `Simpleapplication` (also available on the CD-ROM).
2. Drag and drop a Button control and two Label controls from the Toolbox on the Design view of the Default.aspx page.
3. Set the Text property of the Label1 control to Simple Application Without AJAX, the Button1 control to Display and ID property to Btndisplay.

After setting the properties of the control, the code of the Default.aspx page appears, as shown in Listing 7.1:

Listing 7.1: Showing the Code for the Default.aspx Page

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
  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></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    <asp:Label ID="LabelHadding" runat="server" Text="Simple Application
                Without AJAX"></asp:Label>
         <br />
        <br />
         <asp:Label ID="TimeLabel" runat="server" Text="Click the Button to Display
         Current System Time" ></asp:Label>
         &nbsp;
         <asp:Button ID="Btndisplay" runat="server" Text="Display"
            onclick="Btndisplay_Click" />

    </div>
    </form>
</body>
</html>
```

4. Double-click the Button control (Btndisplay) and modify the code in the Click event of Button, as shown in Listing 7.2:

Listing 7.2: Showing the Code of the Code-Behind File of Default.aspx Page

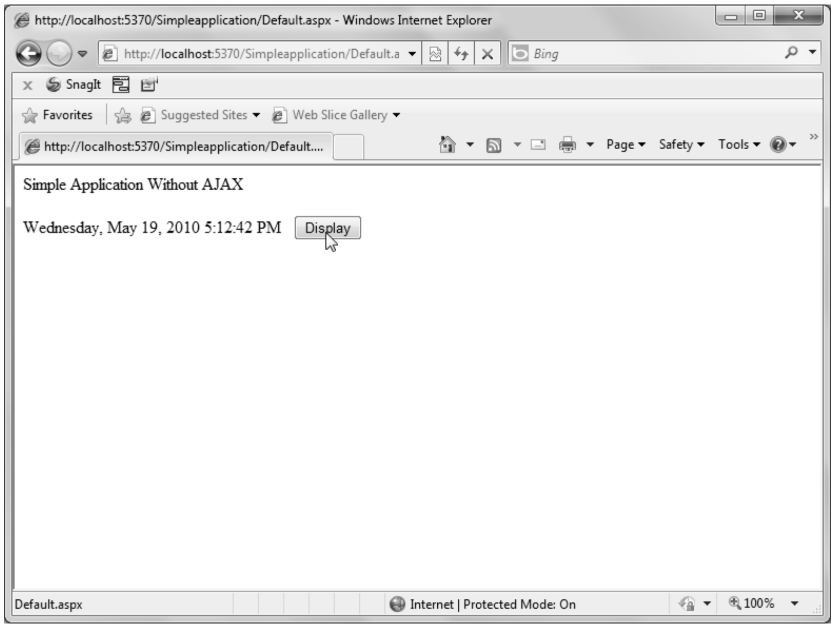
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }

    protected void Btndisplay_Click(object sender, EventArgs e)
    {
        String myDateTime;
        myDateTime = System.DateTime.Today.ToLongDateString() + " ";
        myDateTime += System.DateTime.Now.ToLongTimeString();
        TimeLabel.Text = myDateTime;
    }
}
```

5. Press the F5 key to execute the application. The output of the application appears (Figure 7.6).
6. Click the Display button to view the current time and date on the Web page, as shown in Figure 7.6:


http://localhost:5370/Simpleapplication/Default.aspx - Windows Internet Explorer
Favorites
Suggested Sites
Web Slice Gallery
Page
Safety
Tools
Display
Default.aspx
Internet | Protected Mode: On
100%

Simple Application Without AJAX

Wednesday, May 19, 2010 5:12:42 PM

Figure 7.6: Displaying the Current Date and Time

Creating a Simple AJAX Application

Now, let's perform the following steps to create a simple AJAX ASP.NET website that displays current system time on a button click:

1. Create an ASP.NET website and save it as `simpleAJAXapplication` (also available on the CD-ROM).
2. Drag and drop a Button control and two Label controls from the Toolbox on the Design view of the Default.aspx page.
3. Set the Text property of the Label1 control to Simple Application with AJAX, the Button1 control to Display and ID property to Btndisplay.

After setting the properties of the control, the code of the Default.aspx page appears, as shown in Listing 7.3:

Listing 7.3: Showing the Code for the Default.aspx Page

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
  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></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    <asp:ScriptManager ID="ScriptManager1" runat="server">
                </asp:ScriptManager>
                <asp:UpdatePanel ID="UpdatePanel1" runat="server">
                <ContentTemplate>
                <h2>
                        <asp:Label ID="LabelHadding" runat="server" Text="Simple
                        Application With AJAX"></asp:Label>
                </h2>
                <asp:Label ID="TimeLabel" runat="server" Text="Click the Button to
                Display Current System Time"></asp:Label>
                &nbsp;
                <asp:Button ID="Btndisplay" runat="server" Text="Display"
                   onclick="Btndisplay_Click" />
                </ContentTemplate>
                </asp:UpdatePanel>
    </div>
    </form>
</body>
</html>
```

4. Double-click the Button control (Btndisplay) and modify the code of the Click event of Button, as shown in Listing 7.4:

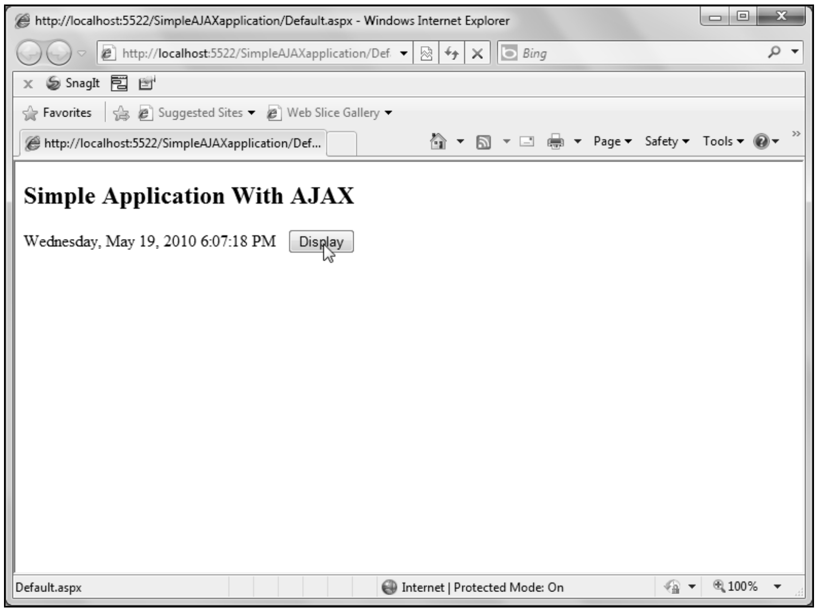
Listing 7.4: Showing the Code of the Code-Behind File of Default.aspx Page

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
    protected void Btndisplay_Click(object sender, EventArgs e)
    {
        String myDateTime;
        myDateTime = System.DateTime.Today.ToLongDateString() + " ";
        myDateTime += System.DateTime.Now.ToLongTimeString();
        TimeLabel.Text = myDateTime;
    }
}
```

5. Press the F5 key to execute the application. The output of the application appears (Figure 7.7).
6. Click the Display button to display the current date and time, as shown in Figure 7.7:


http://localhost:5522/SimpleAJAXapplication/Default.aspx - Windows Internet Explorer
Favorites
Suggested Sites
Web Slice Gallery
Display
Default.aspx
Internet | Protected Mode: On
100%

Simple Application With AJAX

Wednesday, May 19, 2010 6:07:18 PM

Figure 7.7: Displaying the Current Date and Time

Let's now learn how to use the AJAX server controls.

Using AJAX Server Controls

Now, you are familiar with a simple `AJAX` application and have understood the basic idea behind `AJAX`. In this section, you learn to use the following two AJAX Server controls:

- `Timer` control
- `UpdateProgress` control

Let's discuss these controls, one by one.

Timer Control

The `Timer` control is used to update content of the `UpdatePanel` control at predefined intervals. Single `Timer control` can refresh more than one `UpdatePanel` controls simultaneously. The `Timer control` has an `Interval` property that is used to define the time limit after which the content is refreshed. When this interval limit is over, the `Tick` event is raised. You can handle this event to update the content values defined in the `UpdatePanel` control. The `Timer` control can be used in the following two ways:

- Inside the `UpdatePanel` control
- Outside the `UpdatePanel` control

Let's discuss these ways in detail.

Using the Timer Control Inside the UpdatePanel Control

Now, let's perform the following steps to create a simple AJAX ASP.NET website that displays current system time using the `Timer` control inside the `UpdatePanel` control

1. Create an ASP.NET website and save it as `Timercontrolinside` (also available on the CD-ROM).
2. Drag and drop a `ScriptManager` control, `UpdatePanel` control, Timer control and two Label controls from the Toolbox on the Design view of the `Default.aspx` page. To use the `Timer` control inside the `UpdatePanel` Control, drag-and-drop the `Timer` control from the Toolbox as a sub-element of the `<ContentTemplate>` tag.

3. Set the Text property of the Label1 control to Application Using Timer Control, Label2 control to Current System Time, ID property of the Label1 control to LabelHeading, ID property of the Label2 control to TimeLabel, ID property of the Timer control to TimeUpdater, and Interval property to 1000.

After setting the properties of the controls, the code of the Default.aspx page appears, as shown in Listing 7.5:

Listing 7.5: Showing the Code for the Default.aspx Page

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
  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></title>
</head>
<body>
    <form id="form1" runat="server">
    <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="LabelHeading" runat="server" Text="Application Using Timer
         Control"></asp:Label>
         </h2>
         <asp:Label ID="TimeLabel" runat="server" Text="Current System Time"></asp:Label>
         &nbsp;
         </ContentTemplate>
         </asp:UpdatePanel>

    </div>
    </form>
</body>
</html>
```

4. Double-click the `Timer` control (TimeUpdater) in the Design view and add the highlighted code, shown in Listing 7.6, in its `Tick` event:

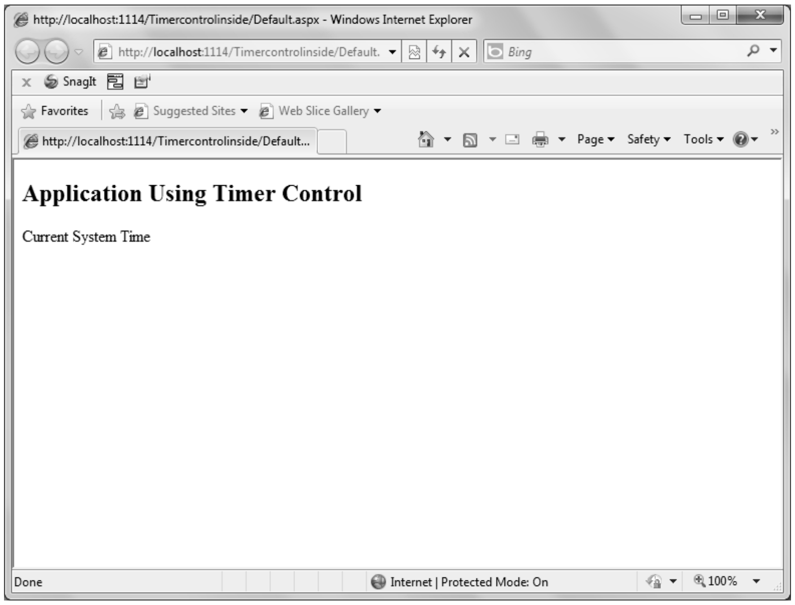
Listing 7.6: Showing the Code of the Code-Behind File of Default.aspx Page

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
    protected void TimeUpdater_Tick(object sender, EventArgs e)
    {
        TimeLabel.Text = System.DateTime.Now.ToLongTimeString();
    }
}
```

5. Press the F5 key to execute the application. The output of the application appears before the Tick event of the Timer control is fired, as shown in Figure 7.8:

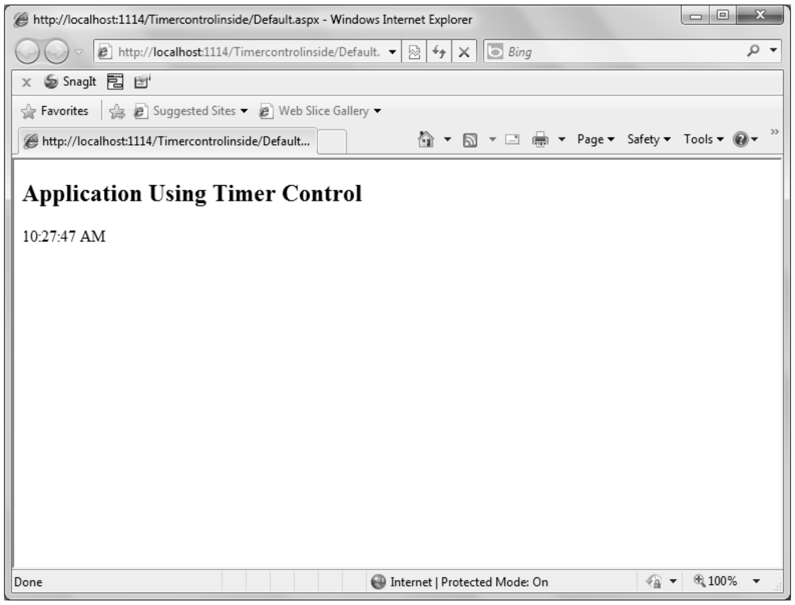

http://localhost:1114/Timercontrolinside/Default.aspx - Windows Internet Explorer
Favorites
Suggested Sites
Web Slice Gallery
Page
Safety
Tools
Done
Internet | Protected Mode: On
100%

Application Using Timer Control

Current System Time

Figure 7.8: Displaying the Output of the Timer Control

After a second, the Tick event is fired and the current system date and time appear, as shown in Figure 7.9:


http://localhost:1114/Timercontrolinside/Default.aspx - Windows Internet Explorer
Favorites
Suggested Sites
Web Slice Gallery
Page
Safety
Tools
Done
Internet | Protected Mode: On
100%

Application Using Timer Control

10:27:47 AM

Figure 7.9: Displaying the Current System Time

Using the Timer Control Outside the UpdatePanel Control

Now, let's perform the following steps to create a simple AJAX ASP.NET website that displays current system time using the Timer control outside the `UpdatePanel` control:

1. Create an ASP.NET website and save it as Timercontroloutside (also available on the CD-ROM).

2. Drag and drop a `ScriptManager` control, `UpdatePanel` control, Timer control outside the `UpdatePanel` control and two `Label` controls from the Toolbox on the Design view of the Default.aspx page.
3. Set the Text property of the Label1 control to Application Using Timer Control, Label2 control to Current System Time, ID property of the Label1 control to LabelHeading, ID property of the Label2 control to TimeLabel, ID property of the Timer control to TimeUpdater, and Interval property to 1000.

After setting the properties of the controls, the code of the Default.aspx page appears, as shown in Listing 7.7:

Listing 7.7: Showing the Code for the Default.aspx Page

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
  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></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    <asp:ScriptManager ID="ScriptManager1" runat="server">
         </asp:ScriptManager>
        <asp:Timer ID="TimeUpdater" runat="server" Interval="1000"
            ontick="TimeUpdater_Tick">
        </asp:Timer>
    <asp:UpdatePanel ID="UpdatePanel1" runat="server">
  <Triggers>
         <asp:AsyncPostBackTrigger ControlID=" TimeUpdater" EventName="Tick" />
  </Triggers>
    <ContentTemplate>
    <asp:Label ID="LabelHeading" 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>
    </form>
</body>
</html>
```

4. Now, double-click the `Timer` control (TimeUpdater) in the Design view and add the highlighted code, shown in Listing 7.8, in its `Tick` event:

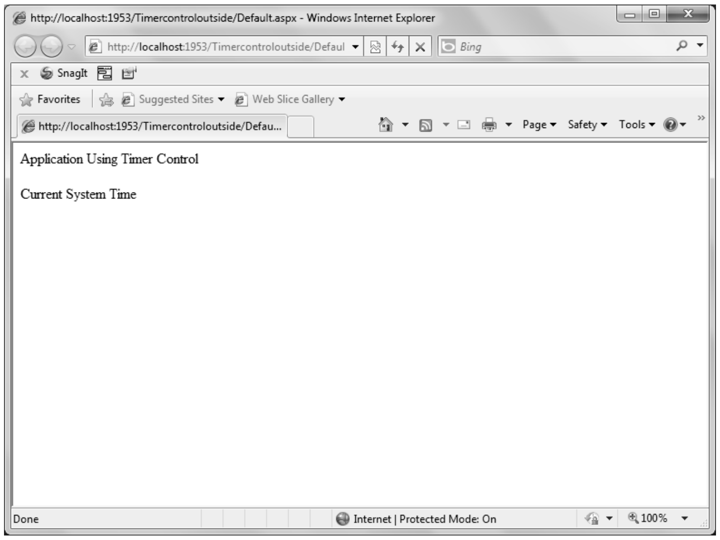
Listing 7.8: Showing the Code of the Code-Behind File of Default.aspx Page

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
    protected void TimeUpdater_Tick(object sender, EventArgs e)
    {
        TimeLabel.Text = System.DateTime.Now.ToLongTimeString();
    }
}
```

5. Press the F5 key to execute the application. The output of the application appears before the Tick event of the Timer control is fired, as shown in Figure 7.10:

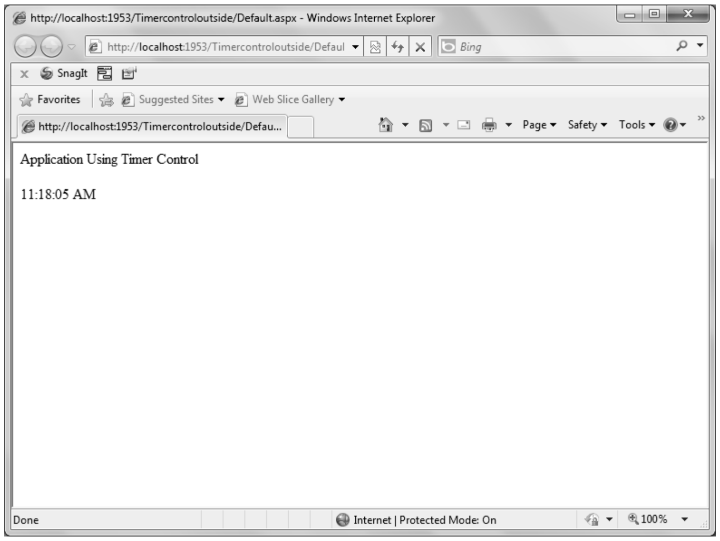

http://localhost:1953/Timercontroloutside/Default.aspx - Windows Internet Explorer
Done
Internet | Protected Mode: On
100%

Application Using Timer Control

Current System Time

Figure 7.10: Displaying the Output

After a second, the Tick event is fired, the current system date and time appears, as shown in Figure 7.11:


http://localhost:1953/Timercontroloutside/Default.aspx - Windows Internet Explorer
Done
Internet | Protected Mode: On
100%

Application Using Timer Control

11:18:05 AM

Figure 7.11: Displaying the Current Date and Time

The UpdateProgress Control

One of the most important advantages of `AJAX` is that it updates the Web page content asynchronously. However, sometimes, this asynchronous updating creates a problem when the updating speed is very slow or data that needs to be updated is of large amount. In such a case, a user does not get any idea whether the request is being processed or not. To resolve this problem, you 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.

Using the UpdateProgress Control for Single UpdatePanel Control

Now, let's create an application named Updateprogress_single to use the `UpdateProgress` control for single `UpdatePanel` control. Perform the following steps to create the Updateprogress_single application:

1. Create an ASP.NET application and save it as Updateprogress_single (also available on the CD-ROM).
2. Drag and drop a ScriptManager control, updatePanel control, and two Label controls from the Toolbox on the Design view of the Default.aspx page.
3. Set the Text property of the Label1 control to Application Using UpdateProgress Control, Label2 control to U**pdateProgress Demo**, ID property of the Label1 control to LabelHeading, ID property of the Label2 control to LabelDemo, and AssociatedUpdatePanelID property of UpdateProgress control to `UpdatePanel1`.

After setting the properties of the controls, the code of the Default.aspx page appears, as shown in Listing 7.9:

Listing 7.9: Showing the Code for the Default.aspx Page

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
  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></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
      <asp:ScriptManager ID="ScriptManager1" runat="server"></asp:ScriptManager>
         <asp:UpdatePanel ID="UpdatePanel1" runat="server">
         <ContentTemplate>
         <asp:UpdateProgress ID="UpdateProgress1" runat="server"
         AssociatedUpdatePanelID="UpdatePanel1">
                 <ProgressTemplate>
                         <strong>
                                 Updation is in progress
                         </strong>
                 </ProgressTemplate>
         </asp:UpdateProgress>
         <h2>
         <asp:Label ID="LabelHeading" runat="server" Text="Application Using
         UpdateProgress Control"></asp:Label>
         </h2>
         <asp:Label ID="LabelDemo" runat="server" Text="UpdateProgress Demo"></asp:Label>
         &nbsp;
         <asp:Button ID="BtnProgress" runat="server" Text="Start Progress"
         onclick="BtnProgress_Click" />
         </ContentTemplate>
         </asp:UpdatePanel>
    </div>
    </form>
</body>
</html>
```

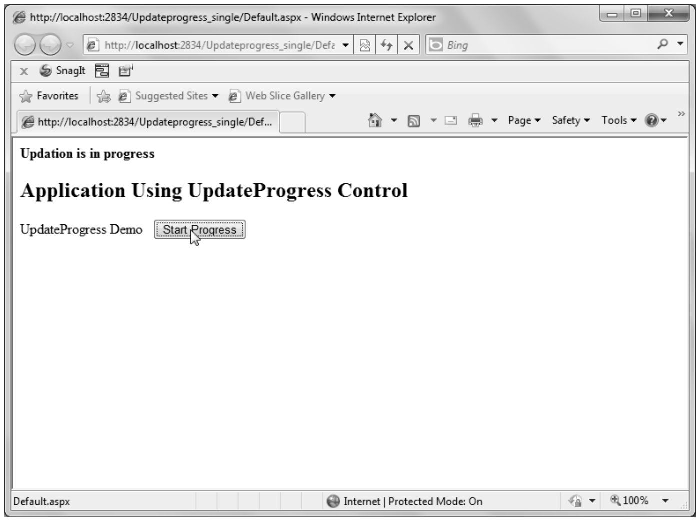
4. Double-click the Button control(BtnProgress) and add the highlighted code, shown in Listing 7.10, in its Click event:

Listing 7.10: Showing the Code of the Code-Behind File of Default.aspx Page

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void BtnProgress_Click(object sender, EventArgs e)
    {
        System.Threading.Thread.Sleep(5000);
        LabelDemo.Text = "Demonstration on UpdateProgress Control taken place last time at
         "+ System.DateTime.Now.ToLongTimeString();
    }
}
```

5. Press the F5 key to execute the application. The output of the application appears (Figure 7.12).
6. Click the Start Progress button (Figure 7.12).

The Updation is in progress message appears, as shown in Figure 7.12.


http://localhost:2834/Updateprogress_single/Default.aspx - Windows Internet Explorer
Start Progress
Default.aspx
Internet | Protected Mode: On
100%

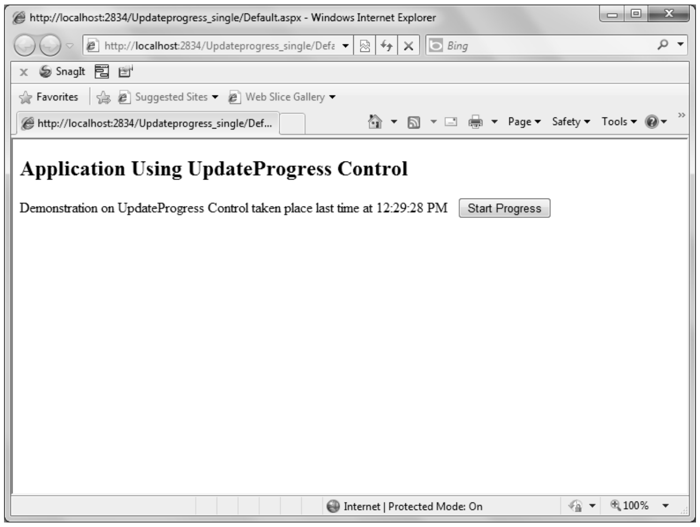
Updation is in progress

Application Using UpdateProgress Control

UpdateProgress Demo

Figure 7.12: Displaying the Output of UpdateProgress Control

When the progress is finished, the system current date and time appears, as shown in Figure 7.13:


http://localhost:2834/Updateprogress_single/Default.aspx - Windows Internet Explorer
Start Progress
Internet | Protected Mode: On
100%

Application Using UpdateProgress Control

Demonstration on UpdateProgress Control taken place last time at 12:29:28 PM

Figure 7.13: Displaying the Progress Finished Message

Using the UpdateProgress Control for Multiple UpdatePanel Controls

Now, let's create an application named Updateprogress_multiple to use the `UpdateProgress` control for multiple `UpdatePanel` controls. Perform the following steps to create the Updateprogress_multiple application:

1. Create an ASP.NET website and save it as Updateprogress_multiple (also available on the CD-ROM).
2. Drag and drop a ScriptManager control, two updatePanel control, and two Label controls from the Toolbox on the Design view of the Default.aspx page.
3. Set the Text property of the Label1 control to Application Using UpdateProgress Control, Label2 control to **UpdateProgress Demo**, ID property of the Label1 control to LabelHeading, ID property of the Label2 control to LabelDemo, and AssociatedUpdatePanelID property of UpdateProgress control to `UpdatePanel1`.

After setting the properties of the controls, the code of the Default.aspx page appears, as shown in Listing 7.11:

Listing 7.11: Showing the Code for the Default.aspx Page

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
  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></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    <asp:ScriptManager ID="ScriptManager1" runat="server"></asp:ScriptManager>
         <asp:UpdateProgress ID="UpdateProgress1" runat="server"
         AssociatedUpdatePanelID="UpdatePanel1">
                 <ProgressTemplate>
                         <strong>
                                 Updation is in progress
                         </strong>
                 </ProgressTemplate>
         </asp:UpdateProgress>
        <asp:UpdatePanel ID="UpdatePanel1" runat="server">
         <ContentTemplate>
        <h2>
         <asp:Label ID="LabelHeading" runat="server" Text="Application Using
         UpdateProgress Control"></asp:Label>
         </h2>
          <asp:Button ID="BtnProgress" runat="server" Text="Start Progress"
         onclick="BtnProgress_Click" />
         &nbsp;
         </ContentTemplate>
         </asp:UpdatePanel>
        <asp:UpdatePanel ID="UpdatePanel2" runat="server">
        <ContentTemplate>
        <asp:Label ID="LabelDemo" runat="server" Text="UpdateProgress Demo"></asp:Label>
        </ContentTemplate>
        </asp:UpdatePanel>
    </div>
    </form>
</body>
</html>
```

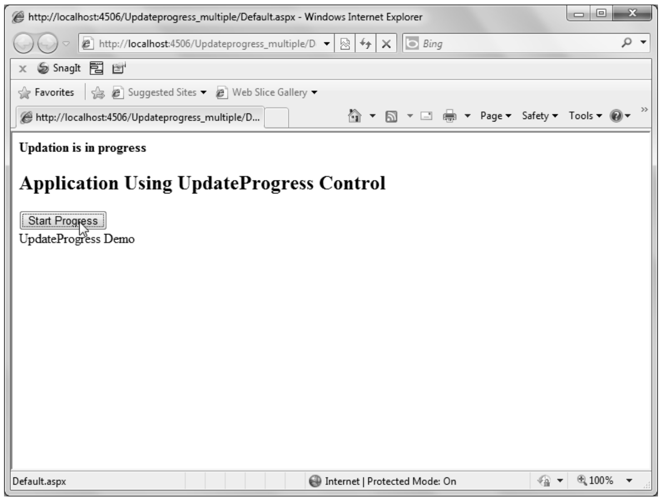
4. Double-click the Button control(BtnProgress) and add the highlighted code, shown in Listing 7.12, in its Click event:

Listing 7.12: Showing the Code of the Code-Behind File of Default.aspx Page

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
```

```
    using System.Web.UI.WebControls;
    public partial class _Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
        }
        protected void BtnProgress_Click(object sender, EventArgs e)
        {
            System.Threading.Thread.Sleep(5000);
            LabelDemo.Text = "Demonstration on UpdateProgress Control taken place last time at
             " + System.DateTime.Now.ToLongTimeString();
        }
    }
```

5. Press the F5 key to execute the application. The output of the application appears (Figure 7.14).
6. Click the Start Progress button (Figure 7.14).The Updation is in progress message set in the UpdatePanel1 control appears, as shown in Figure 7.14:



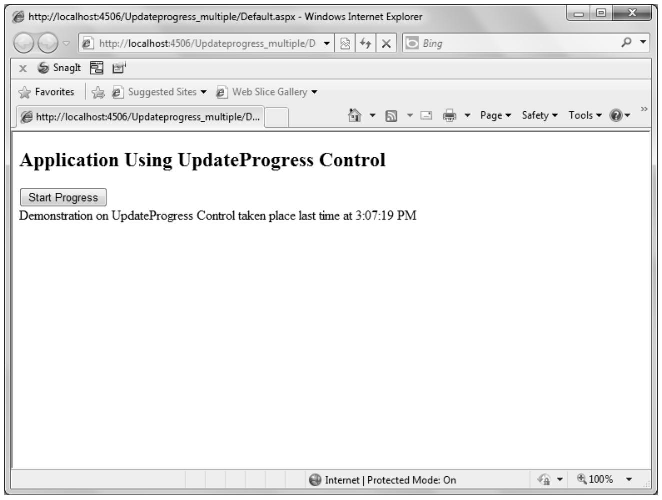
Updation is in progress

Application Using UpdateProgress Control

UpdateProgress Demo

Figure 7.14: Displaying the Output of Multiple UpdatePanel Controls

When the progress is finished, the system current date and time appear, as shown in Figure 7.15:



Application Using UpdateProgress Control

Demonstration on UpdateProgress Control taken place last time at 3:07:19 PM

Figure 7.15: Displaying the Current Date and Time of a System

Let's now learn how to customize AJAX controls.

Customizing AJAX Controls

As discussed, you can customize AJAX controls to add more functionality in your applications. Now, let's create an application named CustomizingAJAX to learn how to customize AJAX controls. Perform the following steps to create the CustomizingAJAX application:

1. Create an ASP.NET website and save it as CustomizingAJAX (also available on the CD-ROM).
2. Add new JavaScript file to the website by right-clicking the website name in Solution Explorer and selecting the Add New Item option. The Add New Item dialog box appears.
3. Select the `JScript File` template from the `Add New Item` dialog box.
4. Modify the code, shown in Listing 7.13, in the JavaScript file(JScript.js):

Listing 7.13: Showing the Code of the JScript.js File

```
Type.registerNamespace("CustomAJAXControl");
// Constructor
CustomAJAXControl.CustomButton = function (element) {
    CustomAJAXControl.CustomButton.initializeBase(this, [element]);
    this._clickDelegate = null;
    this._driftDelegate = null;
    this._undriftDelegate = null;
}
CustomAJAXControl.CustomButton.prototype =
{
    // text property accessors.
    get_text: function () {
        return this.get_element().innerHTML;
    },
    set_text: function (value) {
        this.get_element().innerHTML = value;
    },
    // Bind and unbind to click event.
    add_click: function (handler) {
        this.get_events().addHandler('click', handler);
    },
    remove_click: function (handler) {
        this.get_events().removeHandler('click', handler);
    },
    // Bind and unbind to drift event.
    add_drift: function (handler) {
        this.get_events().addHandler('drift', handler);
    },
    remove_drift: function (handler) {
        this.get_events().removeHandler('drift', handler);
    },
    // Bind and unbind to undrift event.
    add_undrift: function (handler) {
        this.get_events().addHandler('undrift', handler);
    },
    remove_undrift: function (handler) {
        this.get_events().removeHandler('undrift', handler);
    },
    // Release resources before control is disposed.
    dispose: function () {
        var element = this.get_element();
        if (this._clickDelegate) {
            Sys.UI.DomEvent.removeHandler(element, 'click', this._clickDelegate);
            delete this._clickDelegate;
        }
        if (this._driftDelegate) {
            Sys.UI.DomEvent.removeHandler(element, 'focus', this._driftDelegate);
            Sys.UI.DomEvent.removeHandler(element, 'mouseover', this._driftDelegate);
            delete this._driftDelegate;
        }
        if (this._undriftDelegate) {
            Sys.UI.DomEvent.removeHandler(element, 'blur', this._undriftDelegate);
            Sys.UI.DomEvent.removeHandler(element, 'mouseout',
                 this._undriftDelegate);
            delete this._undriftDelegate;
        }
        CustomAJAXControl.CustomButton.callBaseMethod(this, 'dispose');
    },
    initialize: function () {
        var element = this.get_element();
        if (!element.tabIndex) element.tabIndex = 0;
        if (this._clickDelegate === null)
        {
            this._clickDelegate = Function.createDelegate(this, this._clickHandler);
        }
        Sys.UI.DomEvent.addHandler(element, 'click', this._clickDelegate);
        if (this._driftDelegate === null)
         {
            this._driftDelegate = Function.createDelegate(this, this._driftHandler);
        }
        Sys.UI.DomEvent.addHandler(element, 'mouseover', this._driftDelegate);
        Sys.UI.DomEvent.addHandler(element, 'focus', this._driftDelegate);
        if (this._undriftDelegate === null)
        {
            this._undriftDelegate = Function.createDelegate(this,
                 this._undriftHandler);
        }
        Sys.UI.DomEvent.addHandler(element, 'mouseout', this._undriftDelegate);
        Sys.UI.DomEvent.addHandler(element, 'blur', this._undriftDelegate);
        CustomAJAXControl.CustomButton.callBaseMethod(this, 'initialize');
    },
    _clickHandler: function (event) {
        var h = this.get_events().getHandler('click');
        if (h) h(this, Sys.EventArgs.Empty);
    },
    _driftHandler: function (event) {
        var h = this.get_events().getHandler('drift');
        if (h) h(this, Sys.EventArgs.Empty);
    },
    _undriftHandler: function (event) {
        var h = this.get_events().getHandler('undrift');
        if (h) h(this, Sys.EventArgs.Empty);
    }
}
CustomAJAXControl.CustomButton.registerClass('CustomAJAXControl.CustomButton',Sys.UI.Contr
  ol);
if (typeof (Sys) !== 'undefined') Sys.Application.notifyScriptLoaded();
```

5. Add the highlighted code, shown in Listing 7.14, in the Source view of the Default.aspx page:

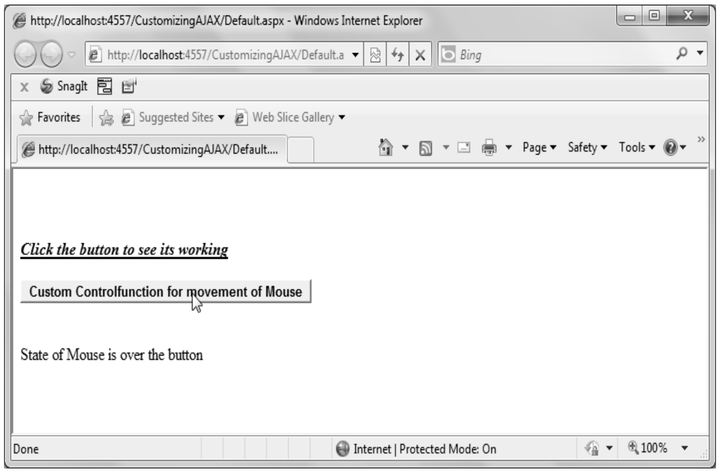
Listing 7.14: Showing the Code for the Default.aspx Page

```
    <%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
      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></title>
    </head>
    <body>
        <form id="form1" runat="server">
        <div>
        <asp:ScriptManager runat="server" ID="ScriptManager01">
                            <scripts>
                                    <asp:ScriptReference Path="~/JScript.js" />
                            </scripts>
                    </asp:ScriptManager>
                    <br />
                    <br />
                    <br />
                    <span style="text-decoration: underline">
                            <em>
```

```
                                <strong>
                         Click the button to see its working
                                        <br />
                                <strong>
                            </em>
                    </span>
                    <br />
                    <script type="text/javascript">
                    var app = Sys.Application;
                    app.add_load(apploadhand);
                    function apploadhand(send, argument)
                    {
                    $create(CustomAJAXControl.CustomButton, {text: 'Custom
                    Controlfunction for movement of Mouse',element: {style:{fontWeight:
                    "bold", borderWidth: "2px"}}}, {click: start,drift:mousehovermovement,
                    undrift: mouseunhovermovement},null,$get('Button1'));
                    }
                    function mousehovermovement(send,argument) {
                        messageonmousemovement = "State of Mouse is over the button"
                        $get('CustomLabel').innerHTML = messageonmousemovement;
                    }
                    function mouseunhovermovement(send,argument) {
                    $get('CustomLabel').innerHTML = "State of Mouse is out of the button";

                    }
                    function start(send, argument)
                    {
                     $get('CustomLabel').innerHTML ="Handling the click event of the Button";
                    }
            </script>
            <button type="button" id="Button1">
            </button>
            &nbsp;<br /><br /><br />
            <div id="CustomLabel">
            </div>
        </div>
        </form>
    </body>
    </html>
```

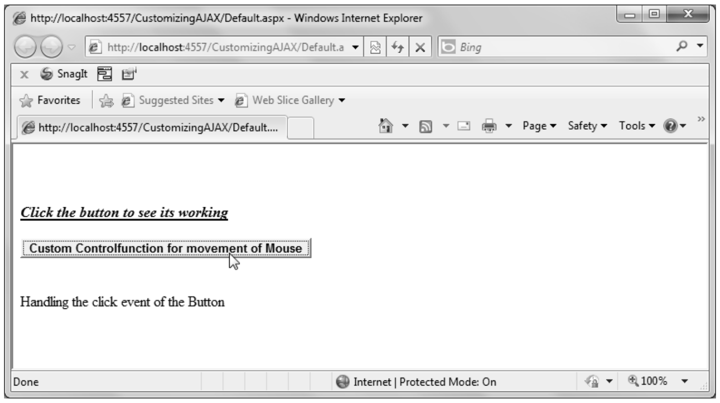
6. Press the F5 key to execute the application.The output of the application appears(Figure 7.16).
7. Move the mouse over the Custom Controlfunction for movement of Mouse button, the message, State of Mouse is over the button, appears on the Web page, as shown in Figure 7.16:


http://localhost:4557/CustomizingAJAX/Default.aspx - Windows Internet Explorer
Click the button to see its working
Custom Controlfunction for movement of Mouse
Done
Internet | Protected Mode: On
100%

State of Mouse is over the button

Figure 7.16:Displaying the Mouseover State

8. Click the Custom Controlfunction for movement of Mouse button (Figure 7.17).The message, Handling the click event of the Button, appears on the Web page, as shown in Figure 7.17:


http://localhost:4557/CustomizingAJAX/Default.aspx - Windows Internet Explorer
Bing
SnagIt
Favorites
Suggested Sites
Web Slice Gallery
Page
Safety
Tools
Done
Internet | Protected Mode: On
100%

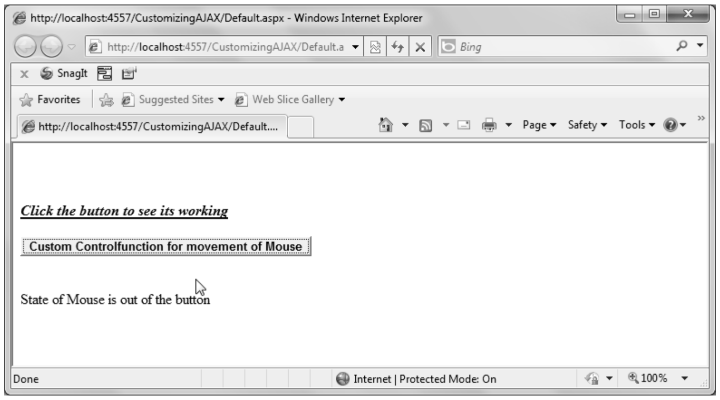
Click the button to see its working

Custom Controlfunction for movement of Mouse

Handling the click event of the Button

Figure 7.17:Handling the Click Event of the Mouse

9. Remove the mouse from the Custom Controlfunction for movement of Mouse button (Figure 7.18).The message, State of Mouse is out of the button, appears on the Web page, as shown in Figure 7.18:


http://localhost:4557/CustomizingAJAX/Default.aspx - Windows Internet Explorer
Bing
SnagIt
Favorites
Suggested Sites
Web Slice Gallery
Page
Safety
Tools
Done
Internet | Protected Mode: On
100%

Click the button to see its working

Custom Controlfunction for movement of Mouse

State of Mouse is out of the button

Figure 7.18: Displaying the Mouseout State

Summary

In this chapter, you have learned about `AJAX, which` calls a server asynchronously and updates a portion of your Web page without refreshing the entire page. The chapter has also discussed the need of implementing AJAX in your applications. You have understood the difference between AJAX and other technologies. Next, you have learned about the new features of AJAX and Microsoft AJAX library. You have also learned about the AJAX architecture, which is based on client-side and server-side architectures. Finally, you have learned about AJAX server controls.

In the next chapter, you will learn about the COM+ applications in C# 2010.

There is a broad list of AJAX enabled custom controls, which are not part of Visual Studio 2010; you have to install them separately. This list is named as `AJAX Control Toolkit`. In the next chapter, you learn about the Working with COM+ Applications.