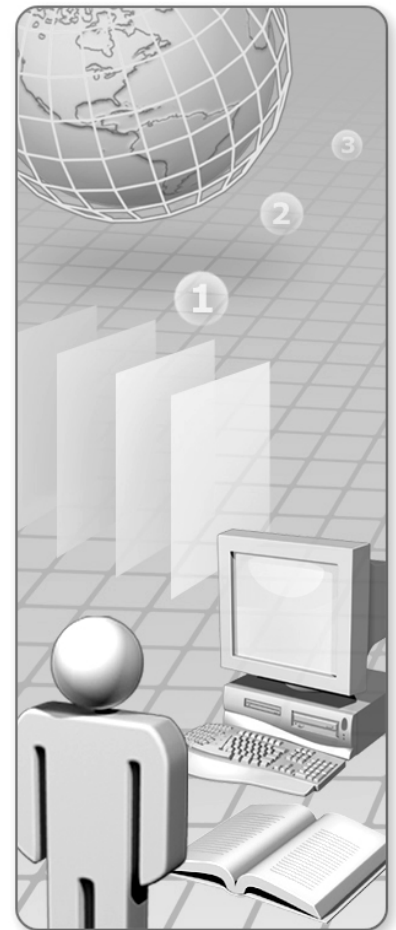


# Unit 3: Optimizing Web Application Performance

---

## Contents

Overview	1
The Page Scripting Object Model	2
Tracing and Instrumentation in Web Applications	4
ASP.NET 2.0 Caching Techniques	6
Asynchronous Processing in Web Applications	8
Web Farm Development Considerations	10
Lab Scenario	12
Lab Tasks and Objectives	13
Lab: Optimizing Web Application Performance	15
Lab Discussion	31



# Overview



- The Page Scripting Object Model
- Tracing and Instrumentation in Web Applications
- ASP.NET 2.0 Caching Techniques
- Asynchronous Processing in Web Applications
- Web Farm Development Considerations
- Lab Scenario
- Lab Tasks and Objectives
- Lab: Optimizing Web Application Performance

## Introduction

Users have little patience with slow Web sites. It is therefore critical that you ensure that your ASP.NET application performs well and responds quickly to user requests. Microsoft® ASP.NET 2.0 includes a variety of facilities to help you achieve this aim.

## Objectives

After completing this unit, you will be able to:

- Describe the Page Scripting Object Model.
- Explain how to use tracing and instrumentation to monitor and improve the performance of a Web application.
- Describe ASP.NET 2.0 caching techniques.
- Explain how asynchronous processing can lead to improved performance for Web applications.
- Describe strategies for dealing with session state management issues when deploying Web applications in a Web farm environment.
- Access Page Scripting Object Model functionality.
- Implement tracing and instrumentation in Web applications.
- Implement ASP.NET 2.0 caching techniques.
- Implement asynchronous processing in Web applications.
- Develop Web applications for Web farm environments.

# The Page Scripting Object Model



- Adding Client Script to ASP.NET Pages
- Client Event Handlers
- Identifying Server Controls in Client Script
- Adding Client Script Dynamically
- Client Callbacks Without Postbacks

## Introduction

ASP.NET is a server-side programming model; code runs on the Web server, usually to formulate the contents of a Web page, before sending that page to the browser on the user's computer as static Hypertext Markup Language (HTML). This means that, to respond to a user action, the user's browser must post the page back across the Internet to the server for processing. It takes time to complete this communication, and this can slow the application significantly. Client-side code can therefore be helpful: if the action can be completed without extra information from the server, client-side code can respond more quickly to the user action.

## Adding Client Script to ASP.NET Pages

If you add client-side script to an ASP.NET page using `<script>` tags, without the **runat="server"** attribute, it will be passed to the browser and will execute there. Although some ASP.NET controls generate their own client script, this does not interfere with any scripts that you have added.

## Client Event Handlers

ASP.NET server controls have a set of attributes defined by their class. If you add an attribute that is not in this set, it will be processed by the browser. You can use this feature to define client-side event handlers for controls. For example, the **TextBox** control has no **onkeyup** attribute. If you add one to a **TextBox** control, it will be processed by the browser. Here it will be interpreted as an event handler for the **onkeyup** event, and the contents will be executed as script.

## Identifying Server Controls in Client Script

If you set the **ID** property for a server control, ASP.NET will render that value as the **ID** and **Name** attributes of the HTML tag that is rendered. You can therefore refer to the control in your client-side code with the same **ID** as in server-side code. If you prefer to use a different identifier on the client side, you can set the **ClientID** property of the control.

## Adding Client Script Dynamically

If the client-side script depends on information that is available only at run time, you cannot add it declaratively at design time. For this situation, ASP.NET enables you to generate and render client-side code at run time. To do this, use the **System.Web.UI.ClientScriptManager** class. The following example adds a script that will execute in the browser when the user attempts to submit the form back to the Web server.

### [Visual Basic]

```
Sub Page_Load()  
    Dim scriptText As String  
    scriptText = "return confirm('Do you want to submit the page?')"  
    ClientScript.RegisterOnSubmitStatement(Me.GetType(), _  
        ConfirmSubmit, scriptText)  
End Sub
```

### [C#]

```
void Page_Load()  
{  
    String scriptText = "return confirm('Do you want to " +  
        "submit the page?')";  
    ClientScript.RegisterOnSubmitStatement(this.GetType(),  
        ConfirmSubmit, scriptText);  
}
```

## Client Callbacks Without Postbacks

Client callbacks provide a way to obtain information from the server without executing the entire page again. They are useful when you have a page that takes a significant time to load and you want to update only a small part of it with information from the server.

In a client callback, client-side code on the page makes a request to the server to execute a page without posting it back. A method on the page can be run and the results returned to the client-side script on the browser to be incorporated into the page. The longer routine associated with reloading the entire page can thus be avoided, and the state data of the controls on the page need not be managed.

# Tracing and Instrumentation in Web Applications



- ASP.NET Tracing and Instrumentation
- .NET Framework Tracing

## Introduction

Developers and testers frequently require information for two purposes: debugging and performance tuning. However, in some circumstances, for example, when the application runs on several different computers simultaneously, it can be difficult to obtain this information. ASP.NET and the Microsoft .NET Framework provide tracing features that enable you to gather this information easily, even in distributed environments.

## ASP.NET Tracing and Instrumentation

To enable tracing for an ASP.NET page, add the attribute **trace="true"** to the `<%@ Page %>` directive. After you have done this, ASP.NET will generate and display diagnostic information at the bottom of the rendered page. Much of this will be relevant to the performance of the application. For example, the size of the view state information in bytes will be displayed.

You can add your own instrumentation data to the trace information displayed by calling methods on the **System.Web.TraceContext** class. In addition to displaying the messages that you send, the trace information will include the times at which the messages were received, which you can use to help diagnose the parts of your code that execute slowly.

## .NET Framework Tracing

The classes in the **System.Diagnostics** namespace provide a separate tracing system. To use this feature, place **Trace** statements at strategic points within your code. Insert the **Trace** statements carefully where crucial information for debugging or optimization is likely to be held in memory.

To receive and store tracing information, use **Tracing Listener** objects. These objects receive tracing output and write it to logs, text files, or the screen.

You can route **System.Diagnostics** tracing information to the browser. This enables you to show all tracing information in the same place. The following example shows you how to send **System.Diagnostics** tracing information to the browser by using the Web.config file:

```
<system.diagnostics>
  <trace>
    <listeners>
      <add name="WebPageTraceListener"
        type="System.Web.WebPageTraceListener,
        System.Web,
        Version=2.0.3600.0,
        Culture=neutral,
        PublicKeyToken=b03f5f7f11d50a3a"/>
    </listeners>
  </trace>
</system.diagnostics>
```

Alternatively, you can send ASP.NET tracing information to a **System.Diagnostics** listener. The following entry in the Web.config file does this:

```
<system.web>
  <trace writeToDiagnosticsTrace="true"/>
  <customErrors mode="Off"/>
</system.web>
```

# ASP.NET 2.0 Caching Techniques



- The Application Cache
- The Page Output Cache
- Partial Page Caching
- Caching Multiple Page Versions

## Introduction

In a Web application, you frequently work with objects or pages that require substantial server resources to create. If you can cache such objects and pages and reuse them, you do not have to re-create them every time a user requests them. In this way, you can improve the performance of your application.

## The Application Cache

The ASP.NET application cache enables you to store objects and data for reuse. This is similar to storing information in the **Application** object. However, objects in the cache are volatile and can be removed without warning, for example, if the server is running low on memory. It is therefore essential to check that an object is available in the cache before attempting to use it.

## The Page Output Cache

A second ASP.NET cache, the page output cache, stores rendered pages in server memory. If two browsers make exactly the same request, the second request will receive a much faster response if the appropriate page is in the page output cache because the page need not be rendered again from scratch. You cache a page by including the `<%@ OutputCache %>` directive on the page, as shown by the following example:

```
<%@ OutputCache Duration="60" VaryByParam="None" %>
```

## Partial Page Caching

In some cases it is not feasible to cache the entire page. For example, a page that includes context-sensitive data should not cache that data. It is possible to cache only parts of the page so that the advantages of caching can still be realized without losing the dynamic nature of your pages.

The first approach you can use is to write a Web user control by including the `<%@ OutputCache %>` directive in this control, rather than on the page. Then only those parts of the page that are within the user control are cached.

Alternatively, you can place the `<%@ OutputCache %>` directive in the page markup, but mark certain sections of the page as exempt from caching. You do this by surrounding these sections with an ASP.NET **Substitution** control.

## Caching Multiple Page Versions

An ASP.NET page frequently responds to user input, so it often generates many different versions of the rendered page. For example, consider a page that displays your company branches close to the user's home. This page will render differently for users in New York than it will for those in San Francisco. You can cache different versions of this page according to the user's location. Use the **varybyparam** attribute in the `<%@ OutputCache %>` directive to do this, as shown in the following example:

```
<%@ OutputCache duration="60" varybyparam="City" %>
```



# Asynchronous Processing in Web Applications



- Setting the Page to Process Asynchronously
- Calling Web Services Asynchronously
- Loading XML Asynchronously

## Introduction

A method that processes or returns a substantial amount of data can take a significant amount of time to complete. Usually the ASP.NET runtime waits for the method to complete before continuing with the next piece of code on a Web page. However, ASP.NET 2.0 enables you to execute some requests asynchronously; you can specify that the runtime should not necessarily wait for a long method to complete before continuing with the next piece of code.

## Setting the Page to Process Asynchronously

To enable code on a Web page to run asynchronously, set the **Page Async** property of the **Page** object. You can do this in the `<%@ Page %>` directive:

```
<%@ Page Async="true" %>
```

To perform an asynchronous operation, you must register two methods: a **Begin** method, which sets the operation running, and an **End** method, which responds when the operation has completed.

## Calling Web Services Asynchronously

Web services can be hosted on servers remote from the one that hosts your Web application, and they can take a long time to execute. Asynchronous execution is useful if you have other operations that do not depend on the results of the Web service. There are two ways to call a Web service asynchronously:

- Use a callback: Create a callback method that matches the **AsyncCallback** delegate. When you call the **Begin** method, you pass the callback method as a parameter. Within the callback method, you then call the **End** method. For more information, refer to the Resource Toolkit document “How to Call Web Services Asynchronously.”
- Use a **WaitHandle**: In this case the **Begin** method returns a **WaitHandle** object. When the Web service method completes, the **WaitHandle.WaitOne** method also completes and the **End** method can be called to use the values returned.

## Loading XML Asynchronously

Another common scenario in which a single operation can take a long time is when you load an Extensible Markup Language (XML) file. Such files can be large and located remotely from the Web server.

You can approach this problem in much the same way as calling a Web service: set the **Async** attribute of the page to true and register **Begin** and **End** methods.

## Web Farm Development Considerations



- Using an External Session State Provider
- Configuring the <machineKey>
- Deploying to a Web Farm

### Introduction

You can improve the response time of your Web application by hosting it on a Web farm. A *Web farm* is a group of servers configured to act as a single Web server and that can share the processing between them. If you have optimized your application fully and upgraded your server as much as possible but the application is still not responding quickly, you should consider moving to a Web farm. If you decide to adopt this approach, you must consider the issues raised here.

### Using an External Session State Provider

By default, information about a user session, such as the user's user name and preferences, is stored in memory on the Web server. However, this is not practical in a Web farm because each request in a session can be directed to different servers in the farm. Therefore, you should use an external session state provider. Two of these are provided with ASP.NET:

- The ASP.NET State Service: This is a separate process, run on a single computer, that maintains the session state. This computer can either be a Web server in the farm or a dedicated computer that does not actually host the application. However, if the computer running the ASP.NET State Service restarts, session state information is lost.
- Microsoft SQL Server™: ASP.NET 2.0 can use a server running SQL Server to store session state information in a database. This is the most robust way to store session state because the session state is saved in a database on the hard disk and is maintained even if the database server or Web servers restart.



**Additional Information** For information on configuring session state storage, see Course 2543, *Core Web Application Technologies with Microsoft Visual Studio 2005*.

---

## Configuring the <machineKey>

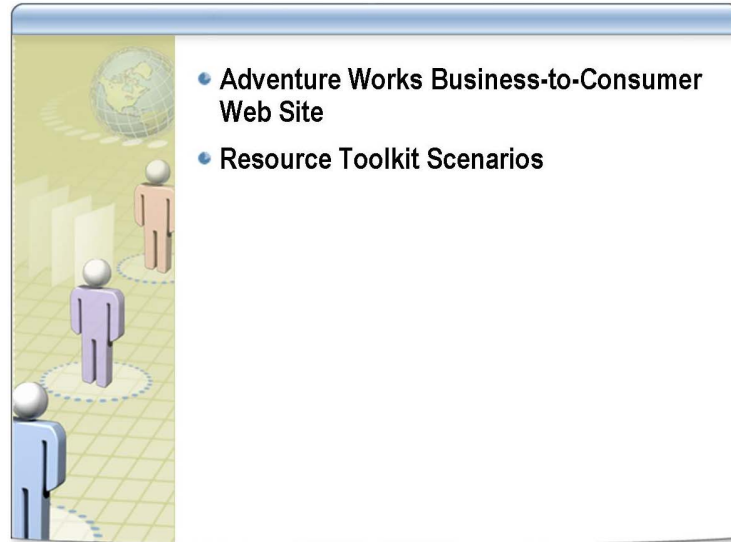
The Machine.config file on each server in the Web farm contains a <machineKey> tag. This tag configures the way encryption is used to protect forms authentication cookies and view state information. The encryption keys are usually generated automatically and are unique to each server. However, in a Web farm, each server must use the same encryption keys so that all servers can access the shared protected information. Refer to the Microsoft Visual Studio® documentation for more information on using the **validationKey** and **decryptionKey** attributes to set these keys.

## Deploying to a Web Farm

Each server in a Web farm must have an identical copy of the application. To ensure consistency, it is helpful to create a Microsoft Windows® Installer package in Visual Studio that you can use to deploy the application.

To do this, add a new Project to your solution by choosing Web Setup Project from the list of available project types. You then add the Web application content, classes, and other files to the Web setup project. To create the Windows Installer package, build the Web setup project.

## Lab Scenario



### Adventure Works Business-to-Consumer Web Site

You are a developer in the Adventure Works organization, a fictitious bicycle manufacturer. You have been asked to assist in creating a new Business-to-Consumer (B2C) Web application and a related Business-to-Employee (B2E) extranet portal.

Decisions on the design of the application have already been made. You have been asked to carry out a number of specific tasks to implement various elements of this design. As part of the B2C development, you have been asked to prototype various performance-related techniques for the Web application.

### Resource Toolkit Scenarios

Before you start work on the lab, you should review the Scenario tab in the Resource Toolkit. The instructor will lead a group discussion of the scenarios for this lab.

## Lab Tasks and Objectives



Task	Objectives
Accessing the Page Scripting Object Model	<ul style="list-style-type: none"> <li>Set the focus on controls</li> <li>Inject client-side script</li> <li>Implement out-of-band callbacks</li> </ul>
Implementing caching	<ul style="list-style-type: none"> <li>Implement page caching</li> <li>Implement parameterized page caching</li> <li>Use substitution controls</li> </ul>
Implementing tracing and instrumentation	<ul style="list-style-type: none"> <li>Add tracing to the Web application</li> <li>Add instrumentation to the Web application</li> </ul>
Implementing asynchronous processing	<ul style="list-style-type: none"> <li>Call Web services asynchronously</li> </ul>

### Lab Tasks

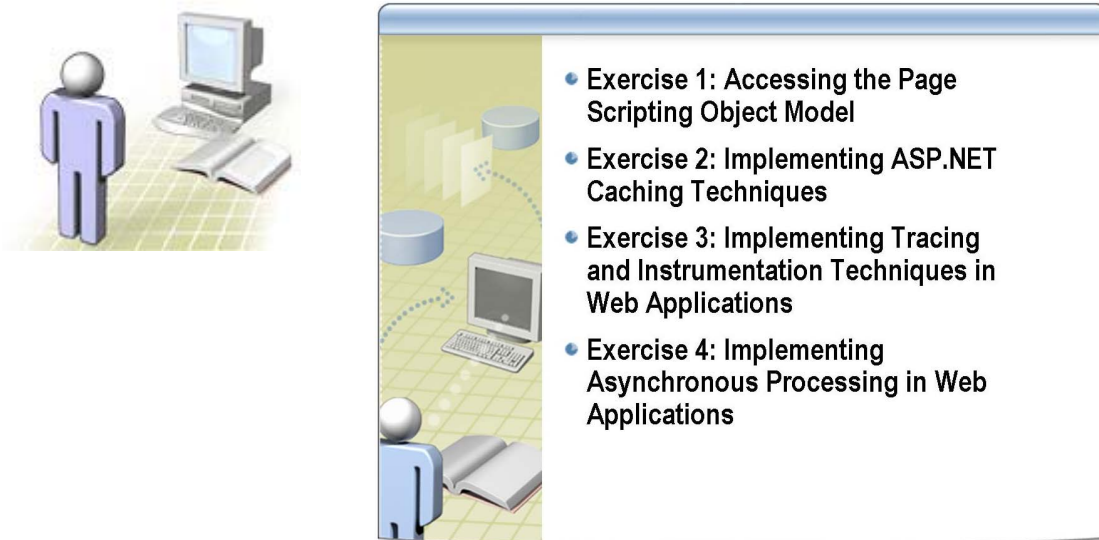
There are four tasks in this lab. Each one is designed to help you achieve one or more learning objectives. Resources are provided in the Resource Toolkit to help you complete the tasks. You should try to complete all of the tasks.

The tasks in this lab are as follows:

- *Accessing the Page Scripting Object Model.* In this task, you will use the Page Scripting Object Model to set the focus on a control. You will then use the **ClientScript** object to inject client-side script onto a Web page. Finally, you will use the **ClientScript** object to implement out-of-band callbacks. The resources for this task are titled:
  - “How to Add Client Script Dynamically to ASP.NET Pages”
  - “Client Callbacks Without Postbacks in ASP.NET Pages”
- *Implementing caching.* In this task, you will implement page caching declaratively. You will then implement parameterized page caching in code. Next, you will prevent portions of a page from being cached by using the **Substitution** control. Finally, you will implement fragment caching by using user controls that have a caching policy different from the page on which they are contained. The resources for this task are titled:
  - “How to Cache ASP.NET Pages”
  - “How to Cache Multiple Versions of a Page”

- *Implementing tracing and instrumentation.* In this task, you will add and review tracing information on a Web page. You will then add event logging capabilities to the Web application, and you will add and manipulate performance counters. Finally, you will test the logging and performance counter management of the Web application. The resources for this task are titled:
  - “How to Implement Tracing Techniques in Web Applications”
  - “How to Log Events in Web Applications”
  - “How to Provide Performance Data for Web Applications”
- *Implementing asynchronous processing.* In this task, you will manage the **Async** property of a page and then add asynchronous event handlers for calling a Web service and receiving data from the Web service. Finally, you will call a Web service asynchronously. The resource for this task is titled “How to Call Web Services Asynchronously.”

## Lab: Optimizing Web Application Performance



After completing this lab, you will be able to:

- Access the Page Scripting Object Model.
- Implement various caching techniques.
- Add tracing and instrumentation to the Web application.
- Call a Web service asynchronously.

Estimated time to complete this lab: **120 minutes**



**Important** You can choose to perform this workshop using either Microsoft Visual Basic® or Microsoft Visual C#®. Code samples and lab solutions are provided in both languages.

### Lab Solutions

There are Visual Basic and Visual C# solution files associated with the labs in this workshop. You can find the lab solution files in the folder E:\Labfiles\Solution on the virtual machines.



## Lab Setup

For this lab you will use the available Microsoft Virtual PC environment.

Before you start the lab, you must do the following:

1. Start the 2544A-LON-DEV-01-03 virtual machine.
2. Log on to the virtual machine with a user name of **Administrator** and a password of **Pa\$\$w0rd**.



**Important** The standard timings for Unit 3 include splitting the lab between day 1 and day 2. If you *do* split the lab in this way, you should ensure that you perform Exercises 1 and 2 at the end of day 1 and then complete Exercises 3 and 4 at the beginning of day 2. If your actual timings do not allow this split, you should complete all lab exercises for Unit 3 either at the end of day 1 or at the beginning of day 2, and the instructor will then adjust the class timings appropriately. Do not split the lab exercises in any other way.

---



**Note** During the lab, please cancel any pop-ups that appear in the virtual machine.

---



## Exercise 1

### Accessing the Page Scripting Object Model

In this exercise, you will use the Page Scripting Object Model to set the focus on a control. You will then use the **ClientScript** object to inject client-side script onto a Web page. Finally, you will use the **ClientScript** object to implement out-of-band callbacks.

Tasks	Supporting information
1. Open the starter solution.	<ol style="list-style-type: none"> <li>If you want to complete this lab by using Visual Basic: <ul style="list-style-type: none"> <li>Open the <b>VB.sln</b> solution in the E:\Labfiles\Starter\VB\ folder.</li> </ul> </li> <li>If you want to complete this lab by using Visual C#: <ul style="list-style-type: none"> <li>Open the <b>CS.sln</b> solution in the E:\Labfiles\Starter\CS\ folder.</li> </ul> </li> </ol> <p>The solution contains a Web application and two Web services.</p>
2. Write and test code to set the focus to the <b>Name</b> text box when displaying the Online Survey page.	<ol style="list-style-type: none"> <li>Run the Web site.</li> <li>When the home page has loaded, use the <b>Contact Us</b> menu to navigate to the <b>Online Survey</b> page. Notice that the focus is not currently set in the <b>Name</b> text box.</li> <li>Close Microsoft Internet Explorer.</li> <li>View the code-behind file for Survey.aspx.</li> <li>Add a method for handling the <b>Page_Load</b> event.</li> <li>Add a decision-making structure to the <b>Page_Load</b> method so that the code it will contain will run only if the page is not being loaded in response to a postback event.</li> <li>Add code to the decision-making structure to set the focus to the <b>txtName</b> control.</li> <li>Run the Web site.</li> <li>When the home page has loaded, use the <b>Contact Us</b> menu to navigate to the <b>Online Survey</b> page. Notice that the focus is now set in the <b>Name</b> text box.</li> <li>In Internet Explorer, on the <b>View</b> menu, click <b>Source</b>. The page source opens in Notepad.</li> <li>Use the Find feature of Notepad to find the following text: <b>WebForm_AutoFocus</b>. The Page Scripting Object Model injected this client-side code onto the Web page to implement the <b>Focus</b> method that you invoked for the <b>txtName</b> control.</li> <li>Note the parameter passed to the <b>WebForm_AutoFocus</b> function call. This is the fully qualified client-side name of the <b>txtName</b> control.</li> <li>Close Notepad and close Internet Explorer.</li> </ol>


*(continued)*

Tasks	Supporting information
<p>3. Use the <b>ClientScript</b> object to inject client-side script that displays a message box onto the Web page.</p>	<p>a. Return to Visual Studio.</p> <p> See the resource “How to Add Client Script Dynamically to ASP.NET Pages” in the Resource Toolkit.</p> <p>b. Add code to the <b>Page_Load</b> method of the Survey.aspx page to perform the following operations:</p> <ul style="list-style-type: none"> <li>• Declare a string variable named <b>sScript</b>, and then initialize it to the following JavaScript code:  <pre>alert('All survey questions are optional. Feel free to omit any that do not apply to you...');</pre> </li> <li>• Invoke the <b>RegisterStartupScript</b> method of the <b>ClientScript</b> object as follows: <ul style="list-style-type: none"> <li>▪ If you are working with Visual Basic, pass in <b>Me.GetType()</b> as the first parameter to the method call.</li> <li>▪ If you are working with Visual C#, pass in <b>this.GetType()</b> as the first parameter to the method call.</li> <li>▪ Pass in the literal string <b>Message</b> as the second parameter.</li> <li>▪ Pass in the <b>sScript</b> variable as the third parameter.</li> <li>▪ Pass in a Boolean value of <b>true</b> as the fourth parameter.</li> </ul> </li> </ul> <p>c. Run the Web site.</p> <p>d. When the home page has loaded, use the <b>Contact Us</b> menu to navigate to the <b>Online Survey</b> page. The client-side alert is displayed.</p> <p>e. Click <b>OK</b>.</p> <p>f. In Internet Explorer, on the <b>View</b> menu, click <b>Source</b>. The page source opens in Notepad.</p> <p>g. Use the Find feature of Notepad to find the following text: <b>alert</b>. The Page Scripting Object Model injected this client-side code onto the Web page to implement the client-side script that you wrote.</p> <p>h. Close Notepad, and then close Internet Explorer.</p>
<p>4. Use the <b>ClientScript</b> object to implement out-of-band callbacks.</p>	<p>a. In Visual Studio, view the markup for the BikeViewer.aspx page.</p> <p>b. Review the page design, taking special notice of the following:</p> <ul style="list-style-type: none"> <li>• The client-side <b>onclick</b> events of the three <b>&lt;div&gt;</b> elements with the text <b>Red</b>, <b>Blue</b>, and <b>Green</b></li> <li>• The <b>ID</b> of the <b>&lt;img&gt;</b> element</li> </ul> <p>c. View the code-behind file for the page.</p> <p> See the resource “Client Callbacks Without Postbacks in ASP.NET Pages” in the Resource Toolkit.</p>


*(continued)*

Tasks	Supporting information
4. <i>(continued)</i>	<p>d. If you are working with Visual Basic, type <b>Implements ICallbackEventHandler</b> on the line beneath the class declaration, and then press ENTER. Ensure that the Visual Studio IDE has created stubs for the following procedures:</p> <ul style="list-style-type: none"> <li>• <b>RaiseCallbackEvent</b></li> <li>• <b>GetCallbackResult</b></li> </ul> <p>e. If you are working with Visual C#, type, <b>ICallbackEventHandler</b> after the class declaration.</p> <p>f. Declare a private string variable named <b>cbresult</b>.</p> <p>g. If you are working with Visual C#, add a public method named <b>RaiseCallbackEvent</b> that accepts a string parameter named <b>eventArgument</b> and does not return any value.</p> <p>h. Add code to the <b>RaiseCallbackEvent</b> method to perform the following actions:</p> <ul style="list-style-type: none"> <li>• If <b>eventArgument</b> is the literal string <b>"Red"</b> or the literal string <b>"Blue"</b>, set the <b>cbresult</b> variable to a concatenation of the following items: <ul style="list-style-type: none"> <li>▪ The literal string <b>"images/"</b></li> <li>▪ The <b>eventArgument</b> variable</li> <li>▪ The literal string <b>"Bike.gif"</b></li> </ul> </li> <li>• If <b>eventArgument</b> is any other value, set the <b>cbresult</b> variable to <b>"File Error"</b>.</li> </ul> <p>i. If you are working with Visual C#, add a public method named <b>GetCallbackResult</b> that returns a string.</p> <p>j. Add a line of code to the <b>GetCallbackResult</b> method to return the <b>cbresult</b> variable.</p> <p>k. In the <b>Page_Load</b> event handler (you must add this method if you are using Visual Basic), declare a string variable named <b>cbRef</b>.</p> <p>l. If you are working with Visual Basic, initialize the <b>cbRef</b> variable to the value returned by the <b>GetCallbackEventReference</b> method of the <b>ClientScript</b> object, passing in the following parameters:</p> <ul style="list-style-type: none"> <li>• The <b>Me</b> object</li> <li>• The literal string <b>"arg"</b></li> <li>• The literal string <b>"ReceiveBikeColor"</b></li> <li>• The literal string <b>"context"</b></li> </ul>

(continued)

Tasks	Supporting information
4. (continued)	<p>m. If you are working with Visual C#, initialize the <b>cbRef</b> variable to the value returned by the <b>GetCallbackEventReference</b> method of the <b>ClientScript</b> object, passing in the following parameters:</p> <ul style="list-style-type: none"> <li>• The <b>this</b> object</li> <li>• The literal string <b>"arg"</b></li> <li>• The literal string <b>"ReceiveBikeColor"</b></li> <li>• The literal string <b>"context"</b></li> </ul> <p>n. Declare a string variable named <b>cbScript</b> and initialize it to a concatenation of the following:</p> <ul style="list-style-type: none"> <li>• The literal string <b>"function GetBikeColor(arg, context) {"</b></li> <li>• The <b>cbRef</b> variable</li> <li>• The literal string <b>"};"</b></li> </ul> <p>o. Invoke the <b>RegisterClientScriptBlock</b> method of the <b>ClientScript</b> object as follows:</p> <ul style="list-style-type: none"> <li>• If you are working with Visual Basic, pass in <b>Me.GetType()</b> as the first parameter to the method call.</li> <li>• If you are working with Visual C#, pass in <b>this.GetType()</b> as the first parameter to the method call.</li> <li>• Pass in the literal string <b>"GetBikeColor"</b> as the second parameter.</li> <li>• Pass in the <b>cbScript</b> variable as the third parameter.</li> <li>• Pass in a Boolean value of <b>true</b> as the fourth parameter.</li> </ul>
5. Add a client-side JavaScript function for receiving the results of the callback.	<p>■ Add the following JavaScript function between the start and end tags of the <b>&lt;Script&gt;</b> element in the markup of the BikeViewer page:</p> <pre>function ReceiveBikeColor(colorFile) {     if(colorFile!="File Error")     {         document.all("imgBike").src = colorFile;         document.all("lblChoose").innerText =             ↳"Choose a color";     }     else     {         document.all("lblChoose").innerHTML =             ↳"Your chosen color is not available.             ↳&lt;BR&gt;Please click a different color";     } }</pre> <p> <b>Note</b> This JavaScript function receives the callback data and updates the images and text on the page accordingly. Note also that this JavaScript might not work in browsers other than Internet Explorer.</p>




(continued)

Tasks	Supporting information
6. Test the callbacks.	<ul style="list-style-type: none"><li>a. Run the Web site.</li><li>b. When the home page has loaded, use the <b>Product</b> menu to navigate to the <b>Bike Viewer</b> page.</li><li>c. Click the <b>Blue</b> link. The image is updated.</li><li>d. Click the <b>Red</b> link. The image is updated.</li><li>e. Click the <b>Green</b> link. The image is not updated, and an appropriate message is displayed.</li><li>f. Close Internet Explorer.</li></ul> <div> <b>Note</b> The updates to the images and text are achieved without performing a full page-request cycle—only the appropriate information is requested and received in an out-of-band manner.</div>




## Exercise 2

### Implementing ASP.NET Caching Techniques

In this exercise, you will implement page caching declaratively. You will then implement parameterized page caching in code. You will then prevent portions of a page from being cached by using the **Substitution** control. Finally, you will implement fragment caching by using user controls that have a caching policy different from the page on which they are contained.



Tasks	Supporting information
1. Implement page caching declaratively.	<p>a. In Visual Studio, view the markup for the Default.aspx page.</p> <p>b. Add the <code>&lt;%@ OutputCache %&gt;</code> directive beneath the <code>&lt;%@ Page %&gt;</code> directive; specify a cache duration of 10 minutes, and then specify that no parameters should be used to cache different versions of the page.</p> <p> See the resource “How to Cache ASP.NET Pages” in the Resource Toolkit.</p>
2. Implement parameterized page caching in code.	<p>a. View the code-behind file for the ProductDetails.aspx page.</p> <p> See the resource “How to Cache Multiple Versions of a Page” in the Resource Toolkit.</p> <p>b. In the method that handles the <b>Page_Load</b> event, add code to perform the following operations:</p> <ul style="list-style-type: none"> <li>Set the <b>ProductID</b> member of the <b>VaryByParams</b> collection of the <b>Response.Cache</b> object to <b>true</b>.</li> <li>Invoke the <b>SetCacheability</b> method of the <b>Response.Cache</b> object, passing in a parameter that specifies that the page can be cached on the server and on the client.</li> <li>Invoke the <b>SetExpires</b> method of the <b>Response.Cache</b> object so that it expires in 10 minutes from the current time.</li> </ul> <p>c. Press F5 to run the Web application. The home page loads and is cached.</p> <p> <b>Note</b> Subsequent requests to the home page will be served from the cache. Although it is difficult to notice any performance improvements in the lab environment, this will make the Web application significantly more efficient when many users are visiting the site concurrently.</p> <p>d. On the <b>Products</b> menu, click <b>View Products</b>.</p> <p>e. In the <b>Categories</b> drop-down list, click <b>Bikes</b>.</p> <p>f. Click [&gt;] for the <b>Mountain Bikes</b> item.</p> <p>g. Click [&gt;] for the <b>Mountain-100 Silver, 38</b> item.</p>

*(continued)*

Tasks	Supporting information
2. <i>(continued)</i>	<p>h. Click <b>[More Details]</b>.</p> <p>i. Examine the address in the Internet Explorer Address bar. Notice that it includes a query string with a <b>ProductID</b> parameter. The code you wrote in this task specified that version of this page should be cached by this parameter.</p> <p> <b>Note</b> Subsequent requests to the ProductDetails page for this specific bike will be served from the cache. A request for a different product will initially not be served from the cache, but then that specific page will also be added to the cache for subsequent requests.</p> <p>j. Close Internet Explorer.</p>
3. Prevent portions of a page from being cached by using a <b>Substitution</b> control.	<p>a. In Visual Studio, view the markup for the Diagnostics.aspx page.</p> <p>b. Add the <code>&lt;%@ OutputCache %&gt;</code> directive beneath the <code>&lt;%@ Page %&gt;</code> directive; specify a cache duration of 10 minutes, and then specify that no parameters will be used to cache different versions of the page.</p> <p>c. Replace the markup for the <b>lblDatabaseStatus</b> label with a <b>Substitution</b> Web server control.</p> <p>d. Set the <b>ID</b> of the <b>Substitution</b> control to <b>subReqs</b>, and then specify that it runs on the server.</p> <p>e. Add a <b>MethodName</b> attribute to the <b>Substitution</b> control with a value of <b>GetDBStatus</b>.</p> <p> <b>Note</b> The method name specifies the method to call when the page is requested, even though the rest of the page might be cached. The method is always called, regardless of the page-caching policy, so it guarantees that the contents of the <b>Substitution</b> control are never cached.</p> <p>f. View the code for the Diagnostics.aspx page.</p> <p>g. Delete the <b>Try</b> and <b>Catch</b> blocks from the <b>Page_Load</b> event.</p> <p> <b>Note</b> The code you have just deleted checked the status of the database every time the page was loaded. However, because the page is now being cached, you will add similar code to the method used by the <b>Substitution</b> control to ensure that it is always up-to-date.</p> <p>h. If you are working with Visual Basic, add a public shared method named <b>GetDBStatus</b> that returns a string.</p> <p>i. If you are working with Visual C#, add a public static method named <b>GetDBStatus</b> that returns a string.</p> <p>j. Specify that the <b>GetDBStatus</b> method accepts an <b>HttpContext</b> parameter named <b>ctx</b>.</p> <p>k. If you are working with Visual Basic, add the contents of the following file to the <b>GetDBStatus</b> method:</p> <ul style="list-style-type: none"> <li>• E:\Labfiles\Starter\CodeText\VB_Code.txt</li> </ul>




(continued)

Tasks	Supporting information
3. (continued)	<p>l. If you are working with Visual C#, add the contents of the following file to the <b>GetDBStatus</b> method:</p> <ul style="list-style-type: none"> <li>• E:\Labfiles\Starter\CodeText\CS_Code.txt</li> </ul> <p>m. After the code you have just added, add a line of code to return the <b>retVal</b> variable.</p> <p> <b>Note</b> The contents of the <b>retVal</b> string variable returned by the <b>GetDBStatus</b> method will be displayed by the <b>Substitution</b> control.</p> <p>n. Save all files.</p> <p>o. Run the Web application.</p> <p>p. When the home page has loaded, click the <b>Diagnostics</b> link. The page is cached, except for the information about the database status. This information will be updated independently of the cache policy of the page every time the page is requested.</p> <p>q. Close Internet Explorer.</p>
4. Implement Web page fragment caching by using a Web user control.	<p>a. Add a new Web user control named <b>ctlAbout.ascx</b> to the Web site.</p> <p>b. Open the <b>About.aspx</b> page in <b>Markup</b> view.</p> <p>c. Cut the entire markup <i>inside</i> the <b>Content</b> control to the Clipboard, but do <i>not</i> cut the <b>Content</b> control tags themselves.</p> <p>d. Paste the markup you have just cut into the <b>Source</b> view of the <b>ctlAbout.ascx</b> user control.</p> <p>e. Add the <code>&lt;%@ OutputCache %&gt;</code> directive beneath the <code>&lt;%@ Control %&gt;</code> directive in the <b>ctlAbout.ascx</b> markup; specify a cache duration of 10 minutes, and then specify that no parameters will be used to cache different versions of the page.</p> <p>f. Save all files.</p> <p>g. Switch to the <b>Design</b> view of the <b>About.aspx</b> page.</p> <p>h. Drag the <b>ctlAbout.ascx</b> user control from the Solution Explorer window, and then drop it onto the <b>Content</b> control in the <b>About.aspx</b> page.</p> <p>i. Save all files.</p> <p>j. Run the Web application.</p> <p>k. When the home page has loaded, click the <b>About Us</b> link.</p> <p> <b>Note</b> Subsequent requests to the About Us page have the contents of the user control served from the cache, even if the rest of the content (on the master page) is rebuilt with every request. Although it is difficult to notice any performance improvements in the lab environment, this will make the Web application significantly more efficient when many users are visiting the site concurrently.</p> <p>l. Close Internet Explorer.</p> <p>m. Close Visual Studio 2005.</p>




## Exercise 3

# Implementing Tracing and Instrumentation Techniques in Web Applications



In this exercise, you will add and review tracing information on a Web page. You will then add event logging capabilities to the Web application, and you will add and manipulate performance counters. Finally, you will test the logging and performance counter management of the Web application.

Tasks	Supporting information
1. Add and review tracing information on a Web page.	<p>a. If you want to complete this exercise by using Visual Basic:</p> <ul style="list-style-type: none"> <li>Open the <b>VB.sln</b> solution in the E:\Labfiles\Starter\VB_Part2\ folder.</li> </ul> <p>b. If you want to complete this exercise by using Visual C#:</p> <ul style="list-style-type: none"> <li>Open the <b>CS.sln</b> solution in the E:\Labfiles\Starter\CS_Part2\ folder.</li> </ul> <p>c. In Visual Studio, view the markup for Diagnostics.aspx.</p> <p> See the resource “How to Implement Tracing Techniques in Web Applications” in the Resource Toolkit.</p> <p>d. Add an attribute to the <code>&lt;%@ Page %&gt;</code> directive that enables tracing.</p> <p>e. View the code-behind file for Diagnostics.aspx.</p> <p>f. Add code to the <i>beginning</i> of the <b>Page_Load</b> method to perform the following actions:</p> <ul style="list-style-type: none"> <li>Set the <b>TraceMode</b> property of the <b>Trace</b> object to <b>TraceMode.SortByTime</b>.</li> <li>Invoke the <b>Write</b> method of the <b>Trace</b> object, passing in a literal string of <b>Page_Load is starting</b>.</li> </ul> <p>g. Add code to the <i>end</i> of the <b>Page_Load</b> method to invoke the <b>Write</b> method of the <b>Trace</b> object, passing in a literal string of <b>Page_Load has completed</b>.</p> <p>h. Locate the <b>GetDBStatus</b> method.</p> <p>i. Add a new line immediately <i>above</i> the line that returns the <b>retVal</b> variable. On this line, add code to invoke the <b>Warn</b> method of the <b>Trace</b> property of the <b>ctx</b> object. Pass in a concatenation of the following items to the method call:</p> <ul style="list-style-type: none"> <li>The literal string <b>Noncached fragment was updated at:</b></li> <li>A space</li> <li><b>DateTime.Now.ToLongTimeString()</b></li> </ul> <p>j. Save all files.</p> <p>k. Press F5 to start the Web services and to run the Web application.</p>

(continued)

Tasks	Supporting information
1. (continued)	<p data-bbox="656 338 1305 401">l. Navigate to the <b>Diagnostics</b> page, and then review the trace information in the page.</p> <p data-bbox="656 411 943 443">m. Close Internet Explorer.</p> <p data-bbox="643 468 704 531"> <b>Note</b> A lot of trace information is displayed. The text <b>Page_Load is starting</b> and <b>Page_Load has completed</b> in the Trace Information section of the output are the results of the code you added to the <b>Page_Load</b> method. The red text displaying the last-updated time of the noncached fragment is the result of the code you added to the <b>GetDBStatus</b> method. Notice that using the <b>Warn</b> method applies the red attributes to the text.</p>
2. Add event logging capabilities to the Web application.	<p data-bbox="656 701 1386 764">a. In Visual Studio, view the code-behind file for the TrailReport.aspx page in the <b>Members</b> folder in Solution Explorer.</p> <p data-bbox="643 789 704 852"> See the resource “How to Log Events in Web Applications” in the Resource Toolkit.</p> <p data-bbox="656 873 1411 957">b. Add a line of code to the <i>beginning</i> of the <b>Page_Load</b> method that declares a <b>DateTime</b> variable named <b>dtmStart</b> and initializes it to the current time.</p> <p data-bbox="656 978 1330 1041">c. Add code to the <i>end</i> of the <b>Page_Load</b> method to perform the following actions:</p> <ul data-bbox="699 1052 1419 1251" style="list-style-type: none"> <li>• Determine whether the event source called <b>TrailReports</b> exists by using the <b>SourceExists</b> method of the <b>System.Diagnostics.EventLog</b> object.</li> <li>• If the source does not exist, use the <b>CreateEventSource</b> method of the <b>System.Diagnostics.EventLog</b> object to create the source called <b>TrailReports</b> in the <b>Adventure-Works</b> log.</li> </ul> <p data-bbox="643 1276 704 1339"> <b>Note</b> Creating event sources is a highly privileged operation that cannot usually be performed from a Web application. However, an event source need only be created once by an Administrator, and then less-privileged code (such as code being run by typical Web site users) can write to that event source.</p> <p data-bbox="727 1444 1495 1728">For this lab, your code will create the event source successfully because you will be running the Web application as the Administrator. However, a more typical approach for ensuring that event sources are created before less-privileged users visit the Web site is to create a Windows Installer class for adding the required event sources and then package the Web application in a Microsoft Installer (MSI) file for deployment. When the application is installed by an Administrator, the event sources will be created.</p> <p data-bbox="656 1749 1403 1812">d. Declare a <b>DateTime</b> variable named <b>dtmEnd</b>, and then initialize it to the current time.</p>

*(continued)*

Tasks	Supporting information
2. <i>(continued)</i>	<p>e. Declare a <b>TimeSpan</b> variable named <b>dtmLoadTime</b>, and then initialize it to the difference between the <b>dtmEnd</b> and <b>dtmStart</b> variables.</p> <p>f. Invoke the <b>WriteEntry</b> method of the <b>System.Diagnostics.EventLog</b> object, passing in the literal string <b>TrailReports</b> as the first parameter. Pass in <b>dtmLoadTime.Milliseconds.ToString()</b> as the second parameter.</p> <p>g. Save all files.</p>
3. Add and manipulate performance counters for the Web application.	<p> See the resource “How to Provide Performance Data for Web Applications” in the Resource Toolkit.</p> <p>a. After the code you added in the previous task, add code to perform the following actions:</p> <ul style="list-style-type: none"> <li>• Determine whether the performance counter category called <b>Adventure-Works</b> exists by using the <b>Exists</b> method of the <b>System.Diagnostics.PerformanceCounterCategory</b> object.</li> <li>• If the performance counter category does not exist, use the <b>Create</b> method of the <b>System.Diagnostics.PerformanceCounterCategory</b> object. Pass in the following parameters to the method: <ul style="list-style-type: none"> <li>▪ The literal string <b>"Adventure-Works"</b></li> <li>▪ The literal string <b>"Adventure-Works Data"</b></li> <li>▪ <b>System.Diagnostics.PerformanceCounterCategoryType.SingleInstance</b></li> <li>▪ The literal string <b>"TrailReports"</b></li> <li>▪ The literal string <b>"Trail report data"</b></li> </ul> </li> </ul> <p> <b>Note</b> Creating performance counters is a highly privileged operation that cannot usually be performed from a Web application. However, a performance counter need only be created once by an Administrator, and then less-privileged code (such as code being run by typical Web site users) can write to that counter.</p> <p>For this lab, your code will create the performance counter successfully because you will be running the Web application as the Administrator. However, a more typical approach for ensuring that performance counters are created before less-privileged users visit the Web site is to create a Windows Installer class for adding the required counters and then package the Web application in an MSI file for deployment. When the application is installed by an Administrator, the performance counters will be created.</p>



*(continued)*

Tasks	Supporting information
3. <i>(continued)</i>	<p>b. Declare a <b>System.Diagnostics.PerformanceCounter</b> variable named <b>trailPerf</b>.</p> <p>c. Initialize the <b>trailPerf</b> variable to a new instance of the <b>System.Diagnostics.PerformanceCounter</b> class, passing in the following parameters:</p> <ul style="list-style-type: none"> <li>• The literal string "<b>Adventure-Works</b>"</li> <li>• The literal string "<b>TrailReports</b>"</li> <li>• The Boolean value <b>false</b></li> </ul> <p>d. Invoke the <b>Increment</b> method of the <b>trailPerf</b> object.</p> <p>e. Save all files.</p>
4. Test the logging and performance counter management of the Web application.	<p>a. Run the Web application.</p> <p>b. Use the <b>Members</b> menu to log in to the Web site as <b>Student</b> with a password of <b>Pa\$\$w0rd</b>.</p> <p>c. Use the <b>Members</b> menu to navigate to the <b>Trail Reports</b> page.</p> <p>d. Leave Internet Explorer running.</p> <p>e. On the <b>Start</b> menu, click <b>Control Panel</b>.</p> <p>f. Double-click <b>Administrative Tools</b>.</p> <p>g. Double-click <b>Event Viewer</b>.</p> <p>h. Click <b>Adventure-Works</b> in the left pane of Event Viewer. A log entry is present in the right pane.</p> <p>i. Double-click the log entry in the right pane, and then review the value it contains. The number represents the amount of time in milliseconds it takes for the <b>Page_Load</b> event to complete. You wrote the code to determine this value, to create the log, and to create the log entry in this exercise.</p> <p>j. Close the <b>Event Properties</b> dialog box, and then close <b>Event Viewer</b>.</p> <p>k. In the <b>Administrative Tools</b> window, double-click <b>Performance</b>.</p> <p>l. Click the <b>Delete</b> button on the toolbar repeatedly until no counters are displayed.</p> <p>m. Click the <b>Add</b> button on the toolbar.</p> <p>n. Click <b>Adventure-Works</b> in the <b>Performance object</b> drop-down list.</p> <p>o. Click <b>TrailReports</b>, and then click <b>Add</b>.</p> <p>p. Click <b>Close</b>. Make a note of the current value of the counter.</p> <p>q. Leave the Performance tool running, and then switch to Internet Explorer.</p> <p>r. Refresh the <b>Trail Reports</b> page.</p> <p>s. Return to the Performance tool, and then note that the counter has been incremented by the value of one. Earlier in this exercise you wrote the code to create the performance counter and to increment its value every time the Trail Reports page is loaded.</p> <p>t. Close the Performance tool, and then close Internet Explorer.</p>


## Exercise 4

### Implementing Asynchronous Processing in Web Applications

In this exercise, you will set the **Async** property of a page and then add asynchronous event handlers for calling a Web service and receiving data from the Web service. Finally, you will test the asynchronous calling of the Web service.

Tasks	Supporting information
1. Enable asynchronous processing on a page.	<ol style="list-style-type: none"> <li>In Visual Studio, view the markup for <code>Diagnostics.aspx</code>.</li> <li>Add an attribute to the <code>&lt;%@ Page %&gt;</code> directive that sets the <b>Async</b> property to <b>true</b>.</li> <li>Add the markup from the following file as the last row of the existing table: <ul style="list-style-type: none"> <li><code>E:\Labfile\Starter\CodeText\Diagnostics_Markup.txt</code></li> </ul> </li> </ol>
2. Add asynchronous event handlers for calling a Web service and receiving data from the Web service.	<ol style="list-style-type: none"> <li>View the code-behind file for the <code>Diagnostics</code> page.</li> <li>Add a class-level <b>TrailReportWebService.Report</b> variable named <b>trailReport</b>, and then instantiate it as a new instance of the <b>TrailReportWebService.Report</b> class.</li> </ol> <p> See the resource “How to Call Web Services Asynchronously” in the Resource Toolkit.</p> <ol style="list-style-type: none"> <li>Add a class-level method named <b>BeginStatus</b> with a return type of <b>IAsyncResult</b>.</li> <li>Specify that the <b>BeginStatus</b> method accepts the following parameters: <ul style="list-style-type: none"> <li>An <b>object</b> parameter named <b>src</b></li> <li>An <b>EventArgs</b> parameter named <b>e</b></li> <li>An <b>AsyncCallback</b> parameter named <b>cb</b></li> <li>An <b>object</b> named <b>state</b></li> </ul> </li> <li>Add code to the <b>BeginStatus</b> method to return the results of invoking the <b>trailReport</b> object’s <b>BeginGetStatus</b> method, passing in the <b>cb</b> object and the <b>state</b> object as parameters.</li> </ol> <p> <b>Note</b> The <b>BeginGetStatus</b> method is a built-in member of objects that represent Web services. It is used when code needs to call the Web service asynchronously. Effectively, a call to this method starts the request for a Web service.</p> <ol style="list-style-type: none"> <li>Add a class-level method named <b>EndStatus</b>.</li> <li>Specify that the <b>EndStatus</b> method has no return type and accepts an <b>IAsyncResult</b> parameter named <b>result</b>.</li> <li>Add code to the <b>EndStatus</b> method to declare a string variable named <b>sStatus</b>.</li> </ol>

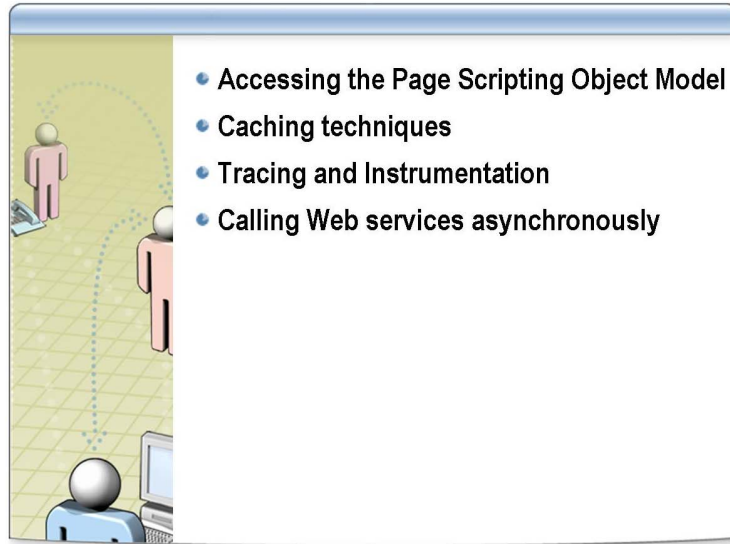
(continued)

Tasks	Supporting information
2. (continued)	<p>i. Initialize <b>sStatus</b> with the result of invoking the <b>EndGetStatus</b> method of the <b>trailReport</b> object, passing in the <b>result</b> variable to the method call.</p> <p> <b>Note</b> The <b>EndGetStatus</b> method is a built-in member of objects that represent Web services. It is used when code needs to call the Web service asynchronously. Effectively, a call to this method receives the response from a Web service that was invoked asynchronously.</p> <p>j. Add a line of code to set the <b>Text</b> property of the <b>lblWebSrvStatus</b> control to the <b>sStatus</b> variable.</p> <p>k. Add code to the <i>beginning</i> of the <b>Page_Load</b> method to perform the following actions:</p> <ul style="list-style-type: none"> <li>• Declare a <b>BeginEventHandler</b> named <b>bh</b>.</li> <li>• If you are working with Visual Basic, initialize the <b>bh</b> variable as a new instance of the <b>BeginEventHandler</b> class, passing in <b>AddressOf Me.BeginStatus</b> as a parameter to the constructor.</li> <li>• If you are working with Visual C#, initialize the <b>bh</b> variable as a new instance of the <b>BeginEventHandler</b> class, passing in <b>this.BeginStatus</b> as a parameter to the constructor.</li> <li>• Declare an <b>EndEventHandler</b> named <b>eh</b>.</li> <li>• If you are working with Visual Basic, initialize the <b>eh</b> variable as a new instance of the <b>EndEventHandler</b> class, passing in <b>AddressOf Me.EndStatus</b> as a parameter to the constructor.</li> <li>• If you are working with Visual C#, initialize the <b>eh</b> variable as a new instance of the <b>EndEventHandler</b> class, passing in <b>this.EndStatus</b> as a parameter to the constructor.</li> <li>• Invoke the <b>AddOnPreRenderCompleteAsync</b> method, passing in the <b>bh</b> and the <b>eh</b> variables as <b>beginHandler</b> and <b>endHandler</b> parameters, respectively.</li> </ul> <p>l. Save all files.</p>
3. Test the asynchronous calling of the Web service.	<p>a. Run the Web application.</p> <p>b. Browse to the <b>Diagnostics</b> page.</p> <p>c. Note the message displaying the availability of the Web service in the <b>Web Service Diagnostics</b> section of the page. The data was retrieved from the Web service in an asynchronous manner by the code you have just written.</p> <p>d. Close Internet Explorer.</p>

## Lab Shutdown

After you complete the lab, you must shut down the 2544A-LON-DEV-01-03 virtual machine and discard any changes.

## Lab Discussion



In the lab, you:

- Accessed the Page Scripting Object Model.
- Implemented various caching techniques.
- Added tracing and instrumentation to the Web application.
- Called a Web service asynchronously.

The instructor will lead a group discussion of these tasks, and you should be prepared to contribute to the discussion, especially if you encountered problems completing the exercises.



