ASP.NET Using C#

Student Guide

Revision 4.6

Object Innovations Course 4140

ASP.NET Using C#
Rev. 4.6

Student Guide

Information in this document is subject to change without notice. Companies, names and data used in examples herein are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Object Innovations.

Product and company names mentioned herein are the trademarks or registered trademarks of their respective owners.

 ™ is a trademark of Object Innovations.

Authors: Robert J. Oberg, Arun Ganesh, Srini Manickam

Special Thanks: Mark Bueno, Peter Thorsteinson, Sharman Staples, Ernani Cecon, Dave Solum, Daniel Berglund

Copyright ©2013 Object Innovations Enterprises, LLC All rights reserved.

Object Innovations
877-558-7246
www.objectinnovations.com

Printed in the United States of America.

Table of Contents (Overview)

Directory Structure

- **The course software installs to the root directory C:\OIC\AspCs.**

  - Example programs for each chapter are in named subdirectories of chapter directories **Chap01**, **Chap02**, and so on.

  - The **Labs** directory contains one subdirectory for each lab, named after the lab number. Starter code is frequently supplied, and answers are provided in the chapter directories.

  - The **CaseStudy** directory contains case studies in multiple steps.

  - The **Demos** directory is provided for doing in-class demonstrations led by the instructor.

- **Data files install to the directory *C:\OIC\Data*.**

- **Log files are written to the directory *C:\OIC\Logs*.**

Table of Contents (Detailed)

Chapter 1

Introduction to ASP.NET

Introduction to ASP.NET

Objectives

After completing this unit you will be able to:

- **Review the fundamentals of Web applications and set up a testbed using Internet Information Services and ASP.NET.**

- **Explain the benefits of ASP.NET.**

- **Describe the programming models provided by ASP.NET: Web Forms, Web services and MVC.**

- **Create a simple Web Forms application using the .NET Framework SDK.**

- **Outline the principal features of ASP.NET.**

Web Application Fundamentals

- **A Web application consists of document and code pages in various formats.**

- **The simplest kind of document is a static HTML page, which contains information that will be formatted and displayed by a Web browser.**

  – An HTML page may also contain hyperlinks to other HTML pages.

  – A hyperlink (or just "link") contains an address, or a Uniform Resource Locator (URL), specifying where the target document is located.

- **The resulting combination of content and links is sometimes called "hypertext" and provides easy navigation to a vast amount of information on the World Wide Web.**

Setting up the Web Examples

- **All the example programs for this chapter are in the chapter folder *Chap01* underneath the root folder *\OIC\AspCs*.**

- **One way to run the examples is to have Internet Information Services (IIS) installed on your system.**

  − You will have to explicitly install it with Windows 7.

- **Appendix C discusses installing and configuring IIS 7.5 in Windows 7.**

- **The management tool for IIS is a Microsoft Management Console (MMC) "snap-in," the Internet Services Manager, which you can find under Administrative Tools in the Control Panel.**

- **You may run also run the ASP.NET examples using the built-in IIS Express.**

  − This built-in Web server is started automatically from within Visual Studio 2013.

  − The use of these Web servers is discussed in Chapter 4

IIS Manager

- **The figure shows the main window of the Internet Services Manager for IIS 7.5, which comes with Windows 7.**

  – We have selected the Content View tab at the bottom of the middle pane.
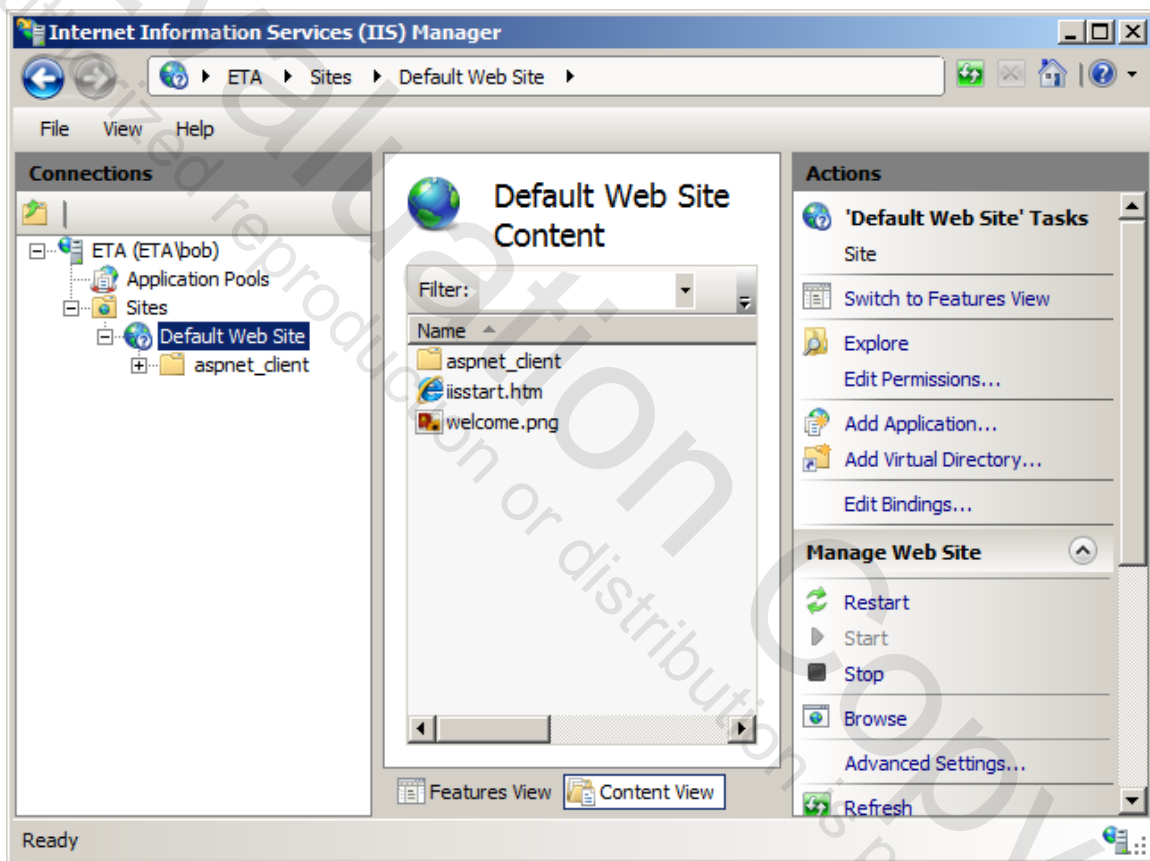
  – You can Start and Stop the Web server and perform other tasks from the Manage Web Site group.

  – With Advanced Settings you can change the physical path of the Default Web Site, which by default is located at **\inetpub\wwwroot** on the drive where Windows is installed.

Virtual Directory

- **You can access Web pages stored at any location on your hard drive by creating a "virtual directory."**

  - Click Add Virtual Directory[1].

  - You can enter the desired alias, which will be the name of the virtual directory.

  - Browse to the desired physical path and click OK.

- **The figure shows creating an alias *AspCs* for the folder *\OIC\AspCs*.**
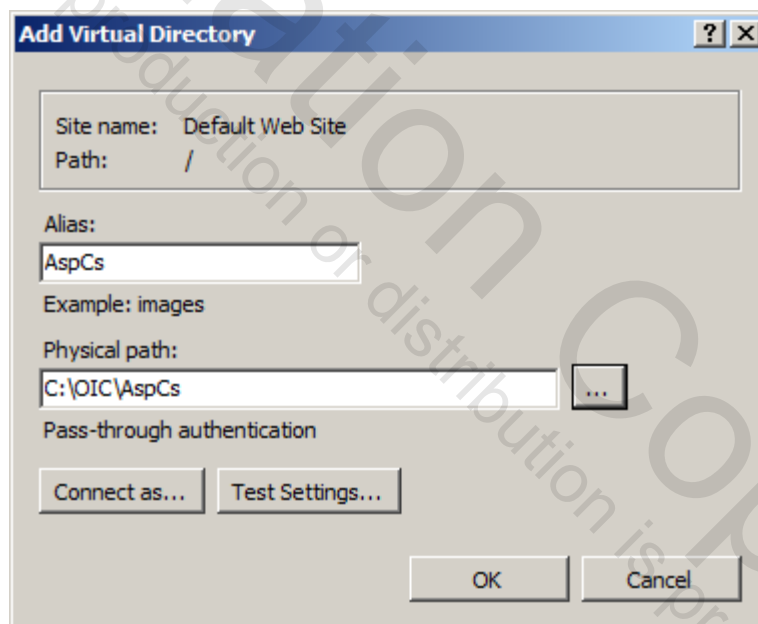
  - You should perform this operation now on your own system so that you may follow along as the course examples are discussed.

---

[1] You can also add a virtual directory from the context menu brought up by a right-click over the Default Web Site.

Home Page for ASP.NET Examples

- **Once a virtual directory has been created, you can access files in it by including the virtual directory in the path of the URL.**

  – In particular, you can access the file **default.htm** using the URL http://localhost/AspCs/.

  – The file **default.htm** contains a home page for all the ASP.NET example programs for the course.
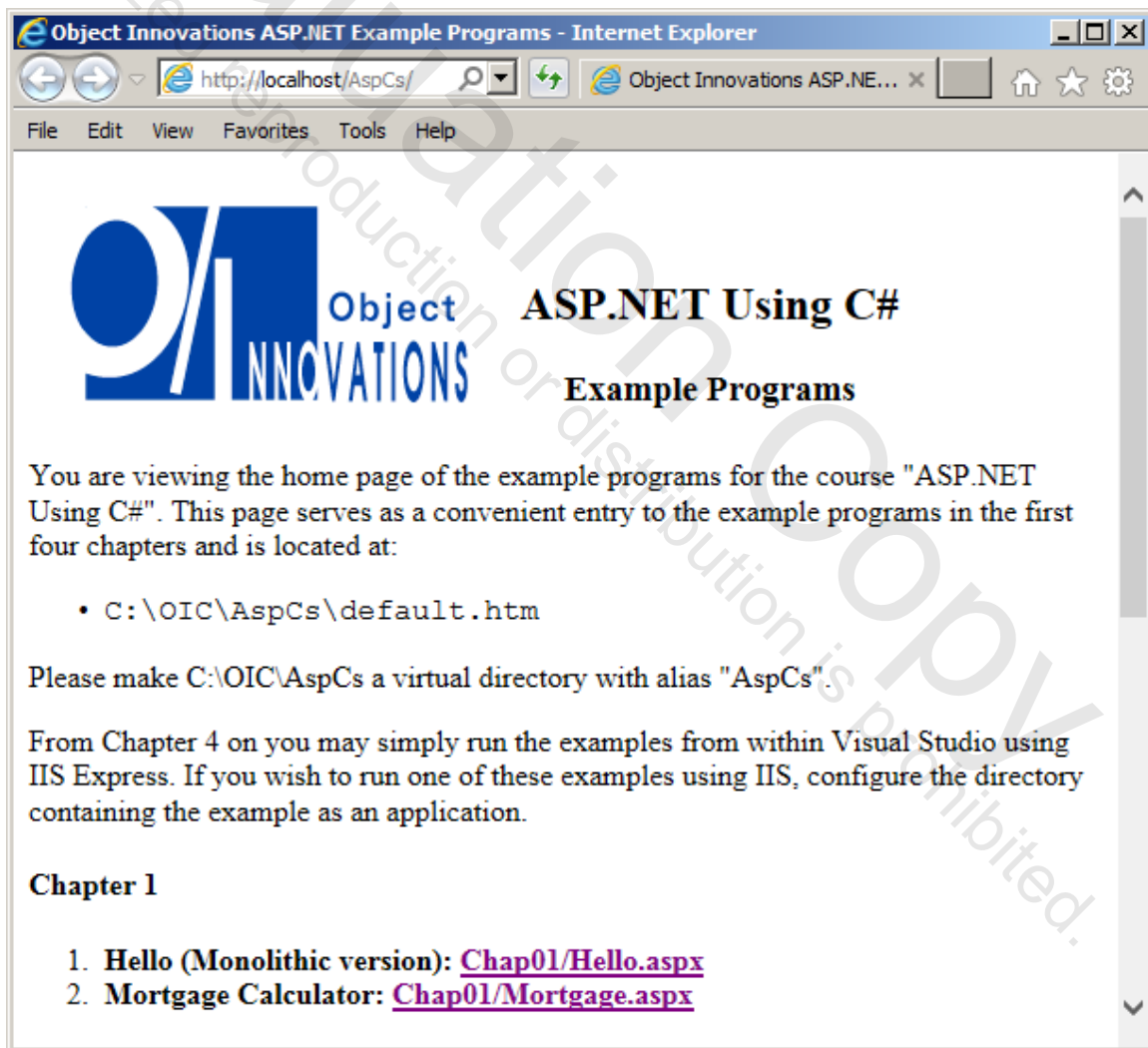
Benefits of ASP.NET

- **You can use compiled, object-oriented languages with ASP.NET, including C# and Visual Basic.**

  – All the power of the .NET Framework is available to you, including the extensive class library.

- **Code and presentation elements can be cleanly separated.**

  – Code can be provided in a separate section of a Web page from user interface elements.

  – The separation can be carried a step further by use of separate "code behind" files.

- **ASP.NET comes with an extensive set of server controls that provide significant functionality out of the box.**

- **Server controls transparently handle browser compatibility issues.**

- **Configuration is handled by XML files without need of any registry settings, and deployment can be done simply by copying files.**

- **Visual Studio provides a very powerful and easy-to-use environment for developing and debugging Web applications.**

ASP.NET Example Program

- ## We examine an example, *Hello.aspx*, in detail.

  - The example is complete in one file and contains embedded server code. ASP.NET pages have an **.aspx** extension.

- ## The source code consists of HTML along with some C# script code. There are also some special tags for "server controls," recognized by ASP.NET.

```
<!-- Hello.aspx -->
<%@ Page Language="C#" %>
<html>
<head>
   <title>Hello</title>
   <script runat="server">
   void cmdEcho_Click(object source,
                      EventArgs e)
   {
      lblGreeting.Text="Hello, " + txtName.Text;
   }
   </script>
</head>
<body>
<form runat="server">Your name: 
<asp:textbox ID="txtName" runat="server">
</asp:textbox>
<p><asp:button ID="cmdEcho" OnClick="cmdEcho_Click"
   Text="Echo" runat="server"
   tooltip="Click to echo your name">
</asp:button></p>
<asp:label ID="lblGreeting"
   runat="server"></asp:label>
</form>
</body>
</html>
```

An Echo Program

- **You can run the program using the URL**
  http://localhost/AspCs/Chap01/Hello.aspx **or by**
  clicking on the link *Chap01/Hello.aspx* **in the home**
  page of the examples programs.

  − The page shows a text box where you can type in your name,
    and there is an "Echo" button.

  − Clicking the button will echo your name back, with a "Hello"
    greeting.

  − The simple form is again displayed, so you could try out
    other names.

  − If you slide the browser's mouse cursor over the button, you
    will see the tool tip "Click to echo your name" displayed in a
    yellow box.

An Echo Program (Cont'd)

- **The figure illustrates a run of this example.**
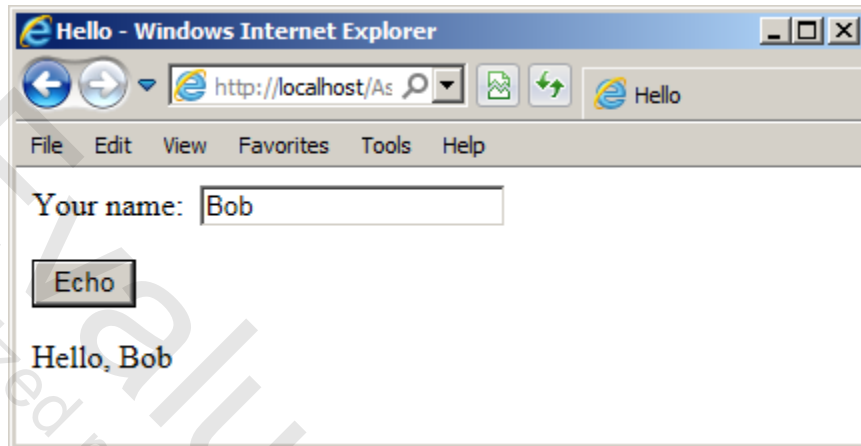
- **This little program would not be completely trivial to implement with other Web application tools, including ASP.**

- **The key user interface feature of such an application is its thoroughly forms-based nature.**

  - The user is presented with a form and interacts with the form.

  - The server does some processing, and the user continues to see the same form.

  - This UI model is second nature in desktop applications but is not so common in Web applications.

  - Typically, the Web server will send back a different page.

An Echo Program (Cont'd)

- **This kind of application could certainly be implemented using a technology like ASP, but the code would be a little ugly.**

- **The server would need to synthesize a new page that looked like the old page, creating the HTML tags for the original page, plus extra information sent back (such as the greeting shown at the bottom in our echo example).**

  − A mechanism is needed to remember the current data that is displayed in the controls in the form.

- **Another feature of this Web application is that it does some client-side processing too— the Echo button's tooltip displayed in a yellow box is performed by the browser.**

  − Such rich client-side processing can be performed by modern browsers, such as Internet Explorer and Firefox.

- **As can be seen by the example code, with ASP.NET it is very easy to implement this kind of Web application.**

- **We will study the code in detail later.**

  − For now, just observe how easy it is!

ASP.NET Features

- **ASP.NET provides a programming model and infrastructure that facilitates developing new classes of Web applications.**

- **Part of this infrastructure is the .NET runtime and framework.**

- **Server-side code is written in .NET compiled languages.**

- **Three main Web programming models are supported by ASP.NET.**

- **Web Forms helps you build form-based Web pages. A WYSIWYG development environment enables you to drag controls onto Web pages.**

  − Special "server-side" controls present the programmer with an event model similar to what is provided by controls in ordinary Windows programming.

- **ASP.NET MVC is a newer framework that provides an alternative to Web Forms for creating Web applications.**

  − It is based on the Model-View-Controller design pattern.

- **ASP.NET Web API supports the creation of HTTP services.**

- **ASP.NET MVC and ASP.NET Web API are introduced in the last two chapter of this course.**

ASP.NET Features (Cont'd)

- **ASP.NET also supports Web Services, which make it possible for a Web site to expose functionality via an API that can be called remotely by other applications.**

  − Data is exchanged using standard Web protocols and formats such as HTTP and XML, which will cross firewalls.

  − A newer technology, Windows Communication Foundation, is now preferred for implementing Web services, along with Web API for HTTP services.

- **Web Forms, ASP.NET MVC, ASP.NET Web API and ASP.NET Web services can all take advantage of the facilities provided by .NET, such as the compiled code and .NET runtime.**

- **In addition, ASP.NET itself provides a number of infrastructure services, including:**

  − state management

  − security

  − configuration

  − caching

  − tracing

Compiled Code

- **Web Forms can be written in any .NET language that that is compatible with the common language runtime, including C# and Visual Basic.**

  - This code is compiled, and thus offers better performance than ASP pages with code written in an interpreted scripting language such as VBScript.

- **Compilation normally occurs at HTTP request time, and subsequent accesses to the page do not require compilation.**

  - The ASP.NET compilation model is described in the next chapter.

- **All of the benefits, such as a managed execution environment, are available to this code, and of course the entire .NET Framework Class Library is available.**

  - Legacy unmanaged code can be called through the .NET interoperability services.

Server Controls

- **ASP.NET provides a significant innovation known as "server controls." These controls have special tags such as *<asp:textbox>*.**

- **Server-side code interacts with these controls, and the ASP.NET runtime generates straight HTML that is sent to the Web browser.**

  – The result is a programming model that is easy to use and yet produces standard HTML that can run in any browser.

Browser Independence

- **Although the World Wide Web is built on standards, the unfortunate fact of life is that browsers are not compatible and have special features.**

  − A Web page designer then has the unattractive options of either writing to a lowest common denominator of browser, or else writing special code for different browsers.

  − Server controls help remove some of this pain.

- **ASP.NET takes care of browser compatibility issues when it generates code for a server control.**

  − If the requesting browser is upscale, the generated HTML can take advantage of these features, otherwise the generated code will be vanilla HTML.

  − ASP.NET takes care of detecting the type of browser.

Separation of Code and Content

- **Typical ASP pages have a mixture of scripting code interspersed with HTML elements.**

- **In ASP.NET there can be a clean separation between code and presentation content.**

  – The server code can be isolated within a single **<script runat="server"> ... / script>** block or, even better, placed within a "code behind" page.

- **We will discuss "code-behind" pages in the next chapter.**
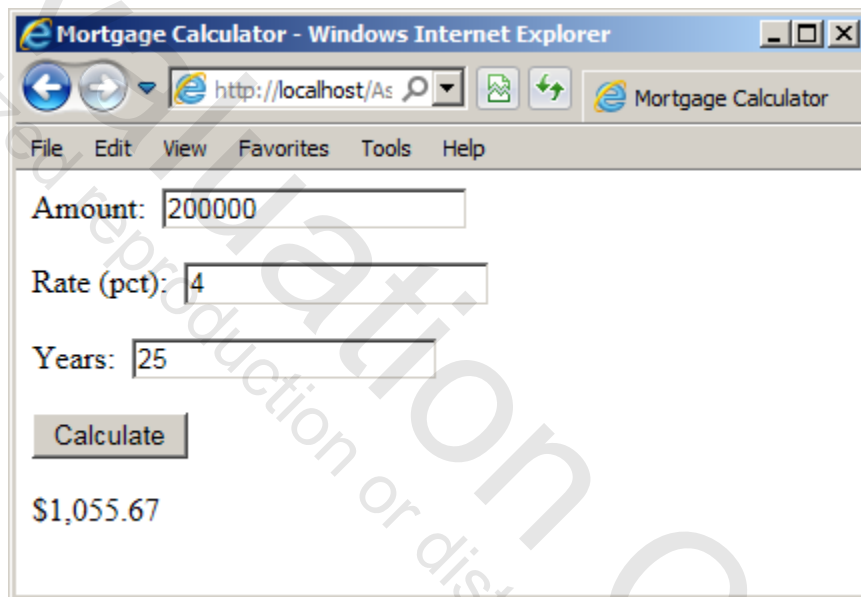
State Management

- **HTTP is a stateless protocol.**

- **Thus, if a user enters information in various controls on a form, and sends this filled-out form to the server, the information will be lost if the form is displayed again, unless the Web application provides special code to preserve this state.**

  – ASP.NET makes this kind of state preservation totally transparent.

  – There are also convenient facilities for managing other types of session and application state.

Lab 1

A Mortgage Calculator Web Page

You have received a consulting contract to add useful functionality to a realtor's Web site. In this lab you will implement the first feature, which is a Web page that can be used to calculate a mortgage payment.

Detailed instructions are contained in the Lab 1 write-up at the end of the chapter.

Suggested time: 30 minutes

Summary

- **ASP.NET is a unified Web development platform that greatly simplifies the implementation of sophisticated Web applications.**

- **ASP.NET supports three Web programming models, Web Forms, ASP.NET MVC and ASP.NET Web API.**

- **Server controls present the programmer with an event model similar to what is provided by controls in ordinary Windows programming.**

- **Other features of ASP.NET include:**

  - Compiled code

  - Browser independence

  - Separation of code and content

  - State management

  - Security

  - Configuration

  - Tracing

  - Caching

Lab 1

A Mortgage Calculator Web Page

Introduction

You have received a consulting contract to add useful functionality to a realtor's web site. In this lab you will implement the first feature, which is a Web page that can be used to calculate a mortgage payment.

The screen capture shows the completed Web page with a sample calculation.
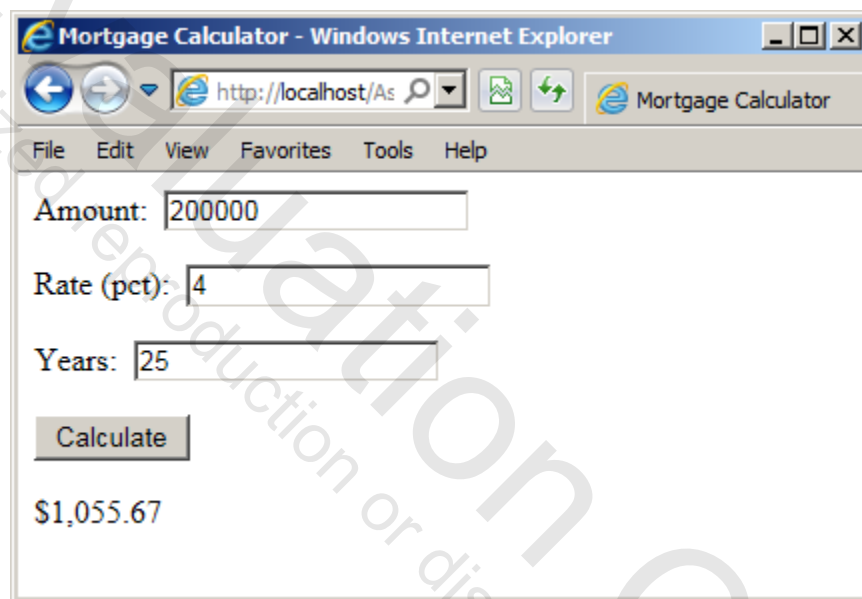
Suggested Time: 30 minutes

Root Directory: **OIC\AspCs**

Directories: **Labs\Lab1**   (do your work here)
                   Chap01\Hello.aspx   (starter file)
                   Chap01\MortgageCalculator   (console mortgage calculator)

Files: **Chap01\Mortgage.aspx**   (answer)

1. As a starter file, copy the file **Hello.aspx** into **Labs\Lab1** and rename as **Mortgage.aspx**. You should now be able to access the "echo" page through the URL **http://localhost/AspCs/Labs/Lab1/Mortgage.aspx**.

2. Change the title to "Mortgage Calculator".

3. Create the user interface for your Web mortgage calculator by making two copies of the textbox for the name in Hello. Rename the three textboxes, one button and one label appropriately. Rename the button handler to match the new name of the button.

4. Examine the code for the C# console program[2] in the **MortgageCalculator** folder. Run this program a couple of times to generate some test data for use in testing your Web mortgage calculator.

5. Implement the event handler for the Calculate button by copying in code from the **MortgageCalculator** project. For the Web version you will obtain a numerical value by using the **Convert** class. For example,

```
decimal amount;      // amount of mortgage
...
amount = Convert.ToDecimal(txtAmount.Text);
```

6. After you have calculated **calcPymt**, display it formatted as currency in the label **lblPayment**. You can do the formatting by the **String.Format** method.

```
lblPayment.Text = String.Format("{0, 8:C}", calcPymt);
```

7. Save your file and test it in the browser. Try out the test data obtained from running the console version of the program.

---

[2] If you are using the free Visual Studio Express 2013 for Web, you will not be able to create Console projects, but you can open and run existing Console projects, which is all you need to do in this lab.

Chapter 2

Web Forms Architecture

Web Forms Architecture

Objectives

After completing this unit you will be able to:

- **Implement an ASP.NET Web application using a code-behind file that separates the visual content from user interface code.**

- **Describe the role of the Page class in Web Forms architecture.**

- **Explain the use of view state to preserve state information in round trips between client and server.**

- **Describe the Web Forms event model and compare it with the Windows Forms event model.**

- **Describe the page directive and outline the common attributes.**

- **Explain how to perform tracing in your ASP.NET applications.**

Web Forms Architecture

- **A Web Form consists of two parts:**

  – The visual content or presentation, typically specified by HTML elements.

  – Code that contains the logic for interacting with the visual elements.

- **A Web Form is physically expressed by a file with the extension *.aspx*.**

- **Any HTML page could be renamed to have this extension and could be accessed using the new extension with identical results to the original.**

  – Thus Web Forms are upwardly compatible with HTML pages.

- **The way code can be separated from the form is what makes a Web Form special.**

  – This code can be either in a separate file (having an extension corresponding to a .NET language, such as **.cs** for C#) or in the **.aspx** file, within a <**script runat="server"> ... </script>** block.

  – When your page is run in the Web server, the user interface code runs and dynamically generates the output for the page.

Code-Behind Version of Echo Example

- **We can understand the architecture of a Web Form most clearly by looking at the code-behind version of our "echo" example.**

  - The visual content is specified by the **.aspx** file **HelloCodebehind.aspx**.

  - The user interface code is in the file **HelloCodebehind.aspx.cs**.

  - Use the link on the home page, or else the URL http://localhost/AspCs/Chap02/Hello/HelloCodebehind.aspx
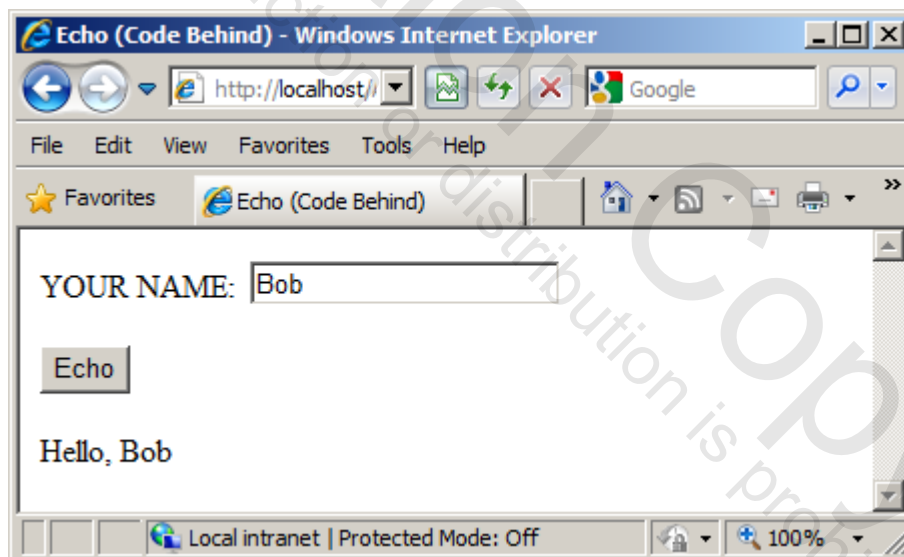
HelloCodebehind.aspx

```
<!-- HelloCodeBehind.aspx -->
<%@ Page Language="C#"
    CodeFile="HelloCodeBehind.aspx.cs"
    Inherits="MyWebPage" %>
<html>
<head>
    <title>Echo (Code Behind)</title>
</head>
<body>
<form runat="server">
  YOUR NAME:  <asp:textbox id="txtName"
    runat="server"></asp:textbox>
  <p><asp:button id="cmdEcho"
    OnClick="cmdEcho_Click" Text="Echo"
    runat="server"></asp:button></p>
  <asp:label id="lblGreeting"
    runat="server"></asp:label>
</form>
</body>
</html>
```

- **This .aspx file specifies the presentation or visual content.**

- **It is HTML with enhancements:**

  - A page directive **<%@ Page Language="C#" ... %>**

  - Special server-side controls such as **<asp:button>**

  - The **runat="server"** attribute

- **This code uses the *CodeFile* attribute to specify the code-behind file.**

HelloCodebehind.aspx.cs

```csharp
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class MyWebPage : System.Web.UI.Page
{
    //protected TextBox txtName;
    //protected Button cmdEcho;
    //protected Label lblGreeting;

    protected void cmdEcho_Click(object sender,
        EventArgs e)
    {
        lblGreeting.Text="Hello, " + txtName.Text;
    }
}
```

- **The .aspx.cs file provides the user interface code.**

  – It is straight C# code.

  – Later we will see examples of other code files that provide additional code processing beyond user interface.

- **Note the use of the *partial* keyword, which enables splitting the definition of a class over two or more source files.**

  – The comments show protected data members representing the controls on the page.

  – These fields are generated automatically at request time.

Page Class

- **The key namespace for Web Forms and Web Services is *System.Web*.**

  − Support for Web Forms is in the namespace **System.Web.UI**.

  − Support for server controls such as text boxes and buttons is in the namespace **System.Web.UI.WebControls**.

  − The class that is used to dynamically generate the output for a Web Form is the **Page** class in the **System.Web.UI** namespace.

- **The next page shows the inheritance model for Page classes used by ASP.NET beginning with .NET 2.0 and employing partial classes.**

Code-Behind Inheritance Model

- **You do not need to define variables for controls.**

    – These variables are defined for you automatically in a *partial class* that is generated from the .aspx file.

```
                      ┌─────────────────────────┐
                      │   System.Web.UI.Page     │
                      └─────────────────────────┘
                                   │
              ┌────────────────────┴────────────────────┐
    ┌───────────────────┐              ┌──────────────────────────┐
    │  My Page Class     │              │  Partial Class Generated │
    │                    │              │  From .aspx File         │
    └───────────────────┘              └──────────────────────────┘
              └────────────────────┬────────────────────┘
                      ┌─────────────────────────┐
                      │   Final Generated Class  │
                      └─────────────────────────┘
```
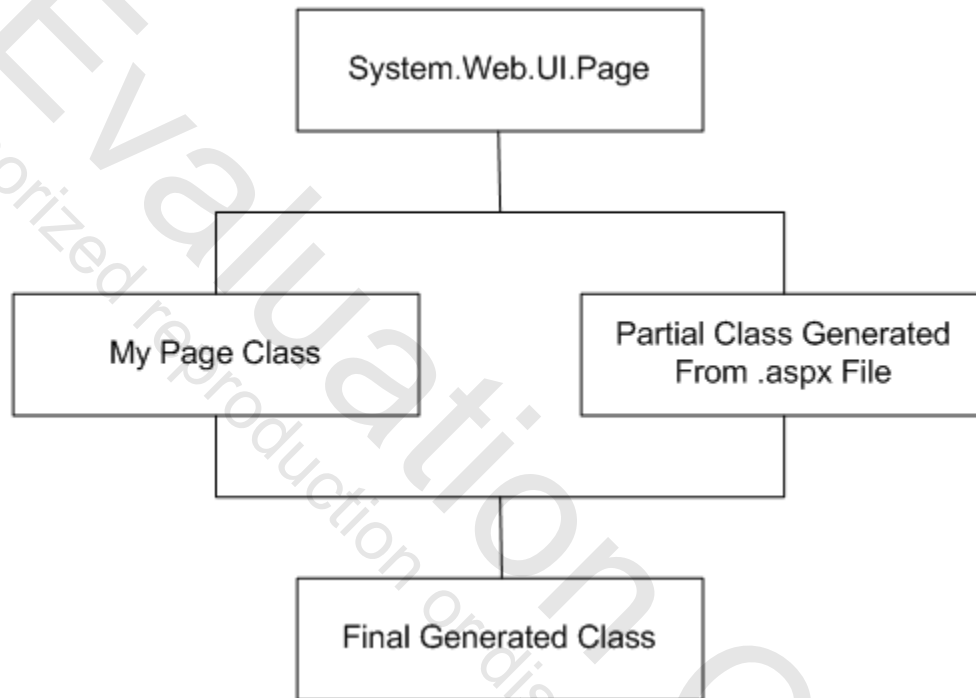
    – Beginning in .NET 2.0 you can split the definition of a class over two or more source files by using the **partial** keyword.

    – At compile time the two partial classes are merged into a single class that inherits from the **Page** class.

    – This class is compiled into an executable, which is run when the page is requested by the browser.

    – The executable code creates the HTML that is sent to the browser.

Web Forms Page Life Cycle

- **We can get a good high-level understanding of the Web Forms architecture by following the life cycle of our simple Echo application.**

- **We will use the code-behind version,** *HelloCodebehind.aspx***.**

1. User requests the **HelloCodebehind.aspx** Web page in the browser.

2. Web server compiles the page class from the **.aspx** file and its associated code-behind page. The Web server executes the code, creating HTML, which is sent to the browser. (In Internet Explorer you can see the HTML code from the menu View | Source.) Note that the server controls are replaced by straight HTML. The code you see is what arrives at the browser, *not the original code on the server*. This compilation occurs on the first reference to the page. Subsequent references use the compiled code.

3. The browser renders the HTML, displaying the simple form shown in the figure. To distinguish this example from the first one, we show "YOUR NAME" in all capitals. Since this is the first time the form is displayed, the text box is empty, and no greeting message is displayed.

Web Forms Page Life Cycle (Cont'd)
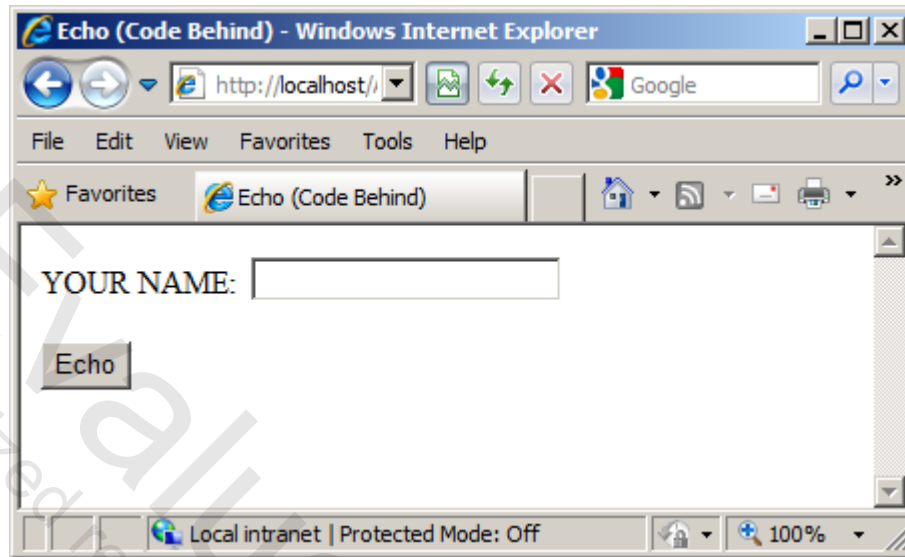
4. The user types in a name (e.g., "Mary Smith") and clicks the "Echo" button. The browser recognizes that a Submit button has been clicked. The method for the form is "post" and the action is "HelloCodebehind.aspx". We thus have what is called a "post back" to the original **.aspx** file.

5. The server now performs processing for this page. An event was raised when the user clicked the "Echo" button, and an event handler in the **MyWebPage** class is invoked.

```
protected void cmdEcho_Click(object Source,
                             EventArgs e)
{
   lblGreeting.Text="Hello, " + txtName.Text;
}
```

Web Forms Page Life Cycle (Cont'd)

6. The **Text** property of the **TextBox** server control **txtName** is used to read the name submitted by the user. A greeting string is composed and assigned to the **Label** control **lblGreeting**, again using property notation.

7. The server again generates straight HTML for the server controls and sends the whole response to the browser.

8. The browser renders the page, as shown in the figure. Now a greeting message is displayed.
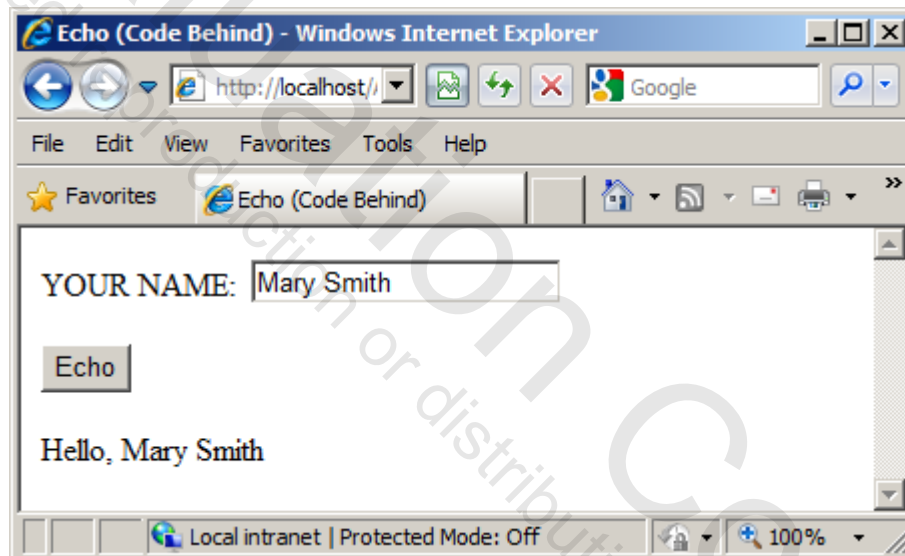
View State

- **An important characteristic of Web Forms is that all information on forms is "remembered" by the Web server.**

  − Since HTTP is a stateless protocol, this preservation of state does not happen automatically but must be programmed.

- **A key feature of ASP.NET Web Forms[1] is that this state information, referred to as "view state," is preserved automatically by the Framework, using a "hidden" control.**

  − This view state is Base 64 encoded and not encrypted.

```
...
<input type="hidden" name="__VIEWSTATE"
value="dDwxMzc4MDMwNTk1O3Q8O2w8aTwyPjs+O2w8dDw7bDxp
PDU+Oz47bDx0PHA8cDxsPFRleHQ7PjtsPEhlbGxvLCBNYXJ5IFN
taXRoOz4+Oz47Oz47Pj47Pj47Pg==" />
...
```

- **Later in the course we will examine other facilities provided by ASP.NET for managing session state and application state.**

---

[1] View state is not employed in ASP.NET MVC.

Enabling View State for Controls

- **By default, view state is enabled for all controls.**

- **But view state increases the amount of time it takes to send a page to the client and post it back.**

  – Storing more view state than is necessary can decrease performance.

- **Beginning with .NET 4.0 you can disable view state for an entire page.**

  – Set the **ViewStateMode** property of the page to **Disabled**.

  – You can then enable view state for just those controls that need it by setting the **EnableViewState** property to true.

- **You could also leave *ViewStateMode* enabled for the page and set *EnableViewState* to false for those controls that do not need it.**

Web Forms Event Model

- **From the standpoint of the programmer, the event model for Web Forms is very similar to the event model for Windows Forms.**

  – Indeed, this similarity is what makes programming with Web Forms so easy.

  – What is actually happening in the case of Web Forms, though, is rather different.

- **The big difference is that events get raised on the client and processed on the server.**

- **Our simple form with one text box and one button is not rich enough to illustrate event processing very thoroughly.**

Web Forms Event Model (Cont'd)

- **Let's imagine a more elaborate form with several text boxes, list boxes, check boxes, buttons, and the like.**

  – Because round trips to the server are expensive, events do not automatically cause a postback to the server.

- **Server controls have what is known as an *intrinsic event set* of events that automatically cause a postback to the server.**

  – The most common such intrinsic event is a button click.

  – Other events, such as selecting an item in a list box, do not cause an immediate postback to the server unless the **AutoPostBack** property of the control is set to true.

  – Otherwise, these events are cached, until a button click causes a post to the server.

  – Then, on the server the various change events are processed, in no particular order, and the event that caused the post is processed.

Page Processing

- **Processing a page is a cooperative endeavor between the Web server, the ASP.NET run-time, and your own code.**

- **The *Page* class provides a number of events, which you can handle to hook into page processing.**

- **The *Page* class also has properties and methods that you can use.**

  - We cover some of the major ones here.

  - For a complete description, consult the .NET Framework documentation.

- **The example programs in this chapter will illustrate features of the *Page* class.**

Page Events

- **A number of events are raised on the server as part of the normal processing of a page.**

- **These events are actually defined in the *Control* base class and so are available to server controls also.**

- **The most important ones are listed below.**

  – **PreInit** is the very first step in the page's life cycle and occurs when the page is initialized. There is no view-state information for any of the controls at this point. This event is is used for loading personalization data and initializing themes. Personalization and themes are discussed later in the course.

  – **Init** is the next step in the page's life. There is still no view-state information for any of the controls at this point.

  – **Load** occurs when the controls are loaded into the page. View-state information for the controls is now available.

  – **PreRender** occurs just before the controls are rendered to the output stream. Normally this event is not handled by a page but is important for implementing your own server controls.

  – **Unload** occurs when the controls are unloaded from the page. At this point it is too late to write your own data to the output stream.

Page Properties

- **The *Page* class has a number of important properties. Some of the most useful are listed below.**

    – **EnableViewState** indicates whether the page maintains view state for itself and its controls. You can get or set this property. The default is **true**, view state is maintained.

    – **ErrorPage** specifies the error page to which the browser should be redirected in case an unhandled exception occurs.

    – **IsPostBack** indicates whether the page is being loaded in response to a postback from the client or is being loaded for the first time.

    – **IsValid** indicates whether page validation succeeded.

    – **Request** gets the HTTP Request object, which allows you to access data from incoming HTTP requests.

    – **Response** gets the HTTP Response object, which allows you to send response data to a browser.

    – **Session** gets the current Session object, which is provided by ASP.NET for storing session state.

    – **Trace** gets a **TraceContext** object for the page, which you can use to write out trace information.

Sample Program

- **We can illustrate some of these features of page processing with a simple extension to our Echo program.**

  – See **HelloPage.aspx**.

- **This page provides handlers for a number of page events, and we write simple text to the output stream, using the *Response* property.**

  – For each event the current text is in the **txtName** and **lblGreeting** server controls. In the handler for **Load** we display the current value of **IsPostBack**, which should be **false** the first time the page is accessed, and subsequently **true**.

Sample Program (Cont'd)

- **When we display the page the first time the output reflects the fact that both the text box and the label are empty, since we have entered no information. *IsPostBack* is false.**

```
Page_Init
txtName =
lblGreeting =
Page_Load
IsPostBack = False
txtName =
lblGreeting =
Page_PreRender
txtName =
lblGreeting =

Your name:  [            ]

[ Echo ]
```
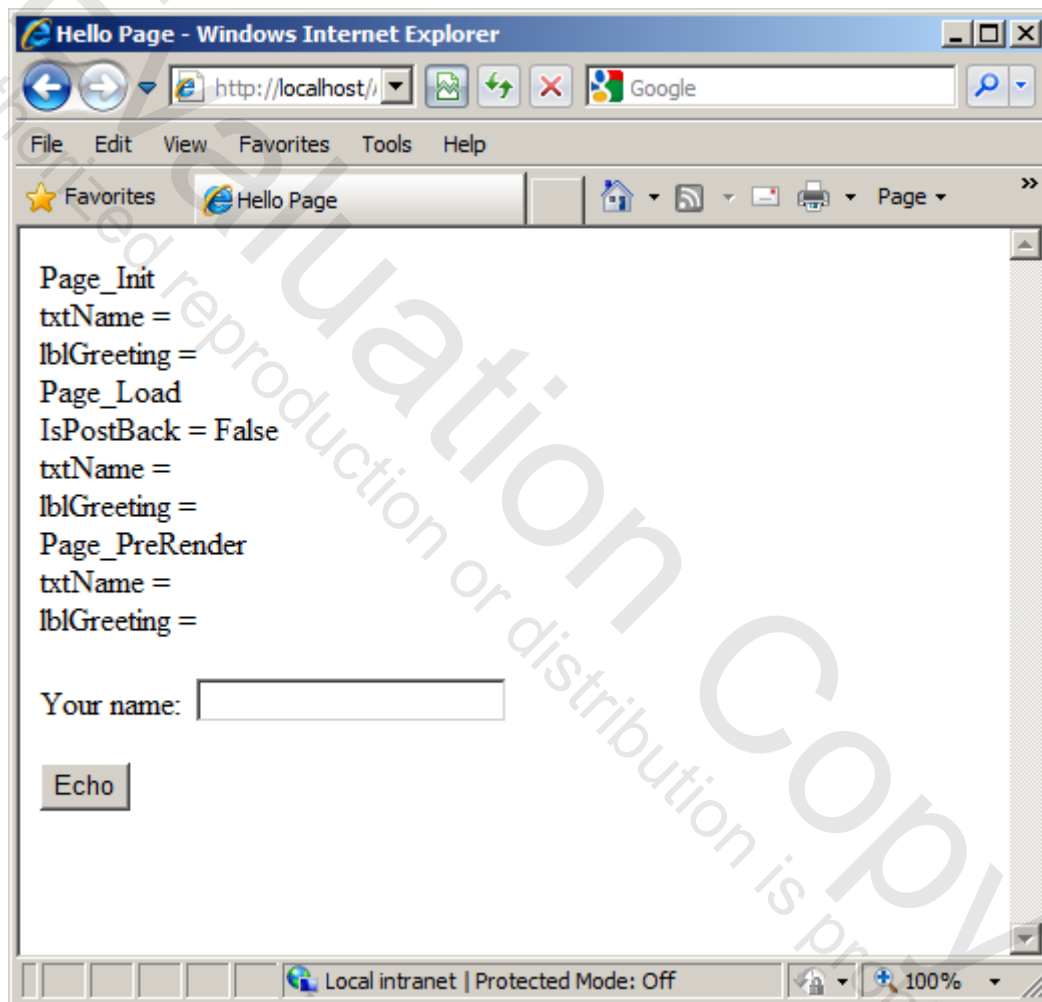
Sample Program (Cont'd)

– Now enter a name and click the "Echo" button. We obtain
   the following output from our handlers for the page events:

```
Page_Init
txtName =
lblGreeting =
Page_Load
IsPostBack = True
txtName = Robert
lblGreeting =
Page_PreRender
txtName = Robert
lblGreeting = Hello, Robert
```

– In **Page_Init** there is no information for either control, since
   view state is not available at page initialization.

– In **Page_Load** the text box has data, but the label does not,
   since the click-event handler has not yet been invoked.
   IsPostBack is now **true**.

– In **Page_PreRender** both controls now have data.

- **Click "Echo" a second time.**

   – Again, the controls have no data in **Page_Init**.

Sample Program (Cont'd)

– This time, however, in **Page_Load** the view state provides data for both controls. The figure shows the browser output after "Echo" has been clicked a second time. Note that **lblGreeting** now has data.
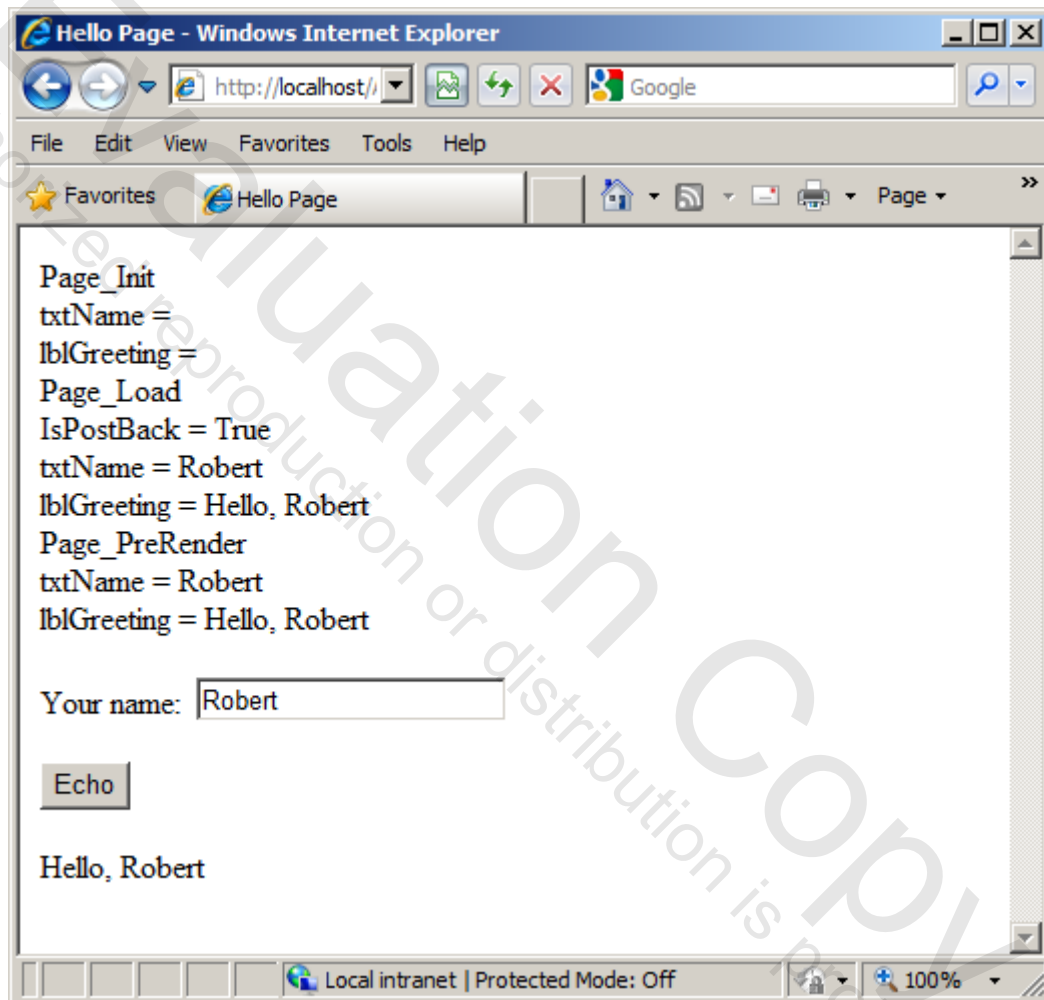
Page Directive

- **An *.aspx* file may contain a page directive defining various attributes that can control how ASP.NET processes the page. A page directive contains one or more attribute/value pairs of the form**

```
attribute="value"
```

within the page directive syntax

```
<%@ Page ... @>
```

- **Our example program *HelloCodebehind.aspx* illustrates an *.aspx* page that does not have any code within it.**

- **The "code-behind" file *HelloCodebehind.aspx.cs* that has the code is specified using the *CodeFile* attribute.**

```
<!-- HelloCodeBehind.aspx -->
<%@ Page Language="C#"
  CodeFile="HelloCodeBehind.aspx.cs"
  Inherits="MyWebPage" %>
<html>
...
```

- **An attribute implicitly used in *HelloPage.aspx* is *AutoEventWireup*, whose default value is true.**

  – **AutoEventWriteup** causes the **Page_Init**, **Page_Load**, **Page_PreRender**, and **Page_Unload** methods to be associated with the **Init**, **Load**, **PreRender**, and **Unload** events.

Page Directive (Cont'd)

- The **CodeFile** attribute identifies the code-behind. (Prior to .NET 2.0 a **Src** attribute was used for this purpose and is still recognized for the sake of compatibility.)

- The **Language** attribute specifies the language used for the page. The code in this language may be in either a code-behind file or a **script** block within the same file. Values can be any ASP.NET-supported language, including C# and Visual Basic.

- The **Inherits** directive specifies the page class from which the page will inherit.

- The **Debug** attribute indicates whether the page should be compiled with debug information. If true, debug information is enabled, and the browser can provide detailed information about compile errors. The default is false.

- The **ErrorPage** attribute specifies a target URL to which the browser will be redirected in the event that an unhandled exception occurs on the page.

- The **Trace** attribute indicates whether tracing is enabled. A value of true turns tracing on. The default is false.

Tracing

- **ASP.NET provides extensive tracing capabilities.**

- **Merely setting the *Trace* attribute for a page to true will cause trace output generated by ASP.NET to be sent to the browser.**

  – In addition, you can output your own trace information using the **Write** method of the **TraceContext** object, which is obtained from the **Trace** property of the **Page**.

- **The page *HelloTrace.aspx* illustrates using tracing in place of writing to the *Response* object.**

Tracing (Cont'd)

- **The figure shows the browser output after entering a name and clicking the Echo button.**

  - Notice that the trace output is shown *after* the form, along with trace information that is generated by ASP.NET itself.

Lab 2

Code-Behind Version of Mortgage Calculator

In this lab you will create a code-behind version of the Mortgage Calculator Web page you implemented in Lab 1. You will study the behavior of this application by tracing methods of the **Page** class.

Detailed instructions are contained in the Lab 2 write-up at the end of the chapter.

Suggested time: 30 minutes

Summary

- **Code-behind files separate the visual content from user interface code.**

- **The *Page* class dynamically generates output for an .aspx page, and your code-behind file contains a class deriving from *Page*.**

- **View state is state information automatically maintained by ASP.NET through a hidden control.**

- **The Web Forms event model is similar to the Windows Form event model, but events typically get raised on the client and processed on the server.**

- **An .aspx file may contain a page directive defining various attributes that can control how ASP.NET processes the page.**

- **You can perform tracing in your ASP.NET applications by turning tracing on in the page directive and writing to the *Trace* object.**

- **The code-behind model uses the attribute *CodeFile* to specify the code-behind file.**

  − Partial classes enable you to write less code in this model.

Lab 2

Code-Behind Version of Mortgage Calculator

Introduction

In this lab you will create a code-behind version of the Mortgage Calculator Web page you implemented in Lab 1. You will study the behavior of this application by tracing methods of the **Page** class.

Suggested Time: 30 minutes

Root Directory: **OIC\AspCs**

Directories: Labs\Lab2   (do your work here)
                 Chap02\Mortgage   (answer)

Part 1. Create Code-Behind Version

1. Copy the file **Mortgage.aspx** from **Labs\Lab1** to use as your starter file. Alternatively, if you did not complete Lab 1 or would like a fresh file, you may copy the answer code for Lab 1 from **Chap01\Mortgage.aspx**.

2. Copy the file **HelloCodebehind.aspx.cs** from **Chap02** and rename to **Mortgage.aspx.cs** to use as your code behind file.

3. Edit the page directive in **Mortgage.aspx** to have the proper **CodeFile** and **Inherits** attributes. You may use **HelloCodehbind.aspx** as a template.

4. Move the C# code from the script block to **Mortgage.aspx.cs** and delete the **script** tags in **Mortgage.aspx**.

5. You should now be able to test via the URL http://localhost/AspCs/Labs/Lab2/Mortgage.aspx.

Part 2. Override Methods of the Page class

1. Using **HelloPage.aspx** as a template, add methods to handle the **Init**, **Load** and **PreRender** methods.

2. Add code to these methods to display the values of your textboxes and label by writing to the **Response** object.

3. Test your program, and make sure you understand the life cycle of a simple ASP.NET web page.

Part 3. Tracing Methods of the Page class

1. In the file **Mortgage.aspx** add the attribute **Trace = true** to the page directive.

2. In the file **Mortgage.aspx.cs** replace the calls to **Response.Write** by calls to **Trace.Write**.

3. Test your program and make sure you understand the basic output of the trace. Don't worry about all the details.

Chapter 9

Debugging, Diagnostics and Error Handling

Debugging, Diagnostics and Error Handling

Objectives

After completing this unit you will be able to:

- **Debug ASP.NET applications created using Visual Studio.**

- **Perform tracing at both the application and page level.**

- **Handle errors in your ASP.NET applications.**

ASP.NET Diagnostics

- **Debugging ASP applications has been notoriously difficult.**

  – A primary debugging tool is **Response.Write**().

- **ASP.NET applications can be debugged the same way as other .NET applications and components.**

  – ASP.NET applications are compiled into executable assemblies, so the same techniques apply.

- **Debugging ASP.NET applications using Visual Studio is easy, because any ASP.NET web site can be opened in Visual Studio.**

- **ASP.NET also provides convenient tracing facilities at both the page and application level.**

- **Later in the chapter we will also look at error handling in ASP.NET.**

Debugging Using Visual Studio

- **Applications created using Visual Studio are very easy to debug.**

  – Build in Debug mode (the default).

  – You can then set breakpoints, single-step, use watch windows, and all the other features of the Visual Studio debugger.

Calculator Example

- **We will illustrate diagnostics in this chapter with a simple Web calculator program *VsCalculator*.**

  – Work with the copy in the **Demos** folder. The original program is in **VsCalculator\Step0** in the chapter folder.
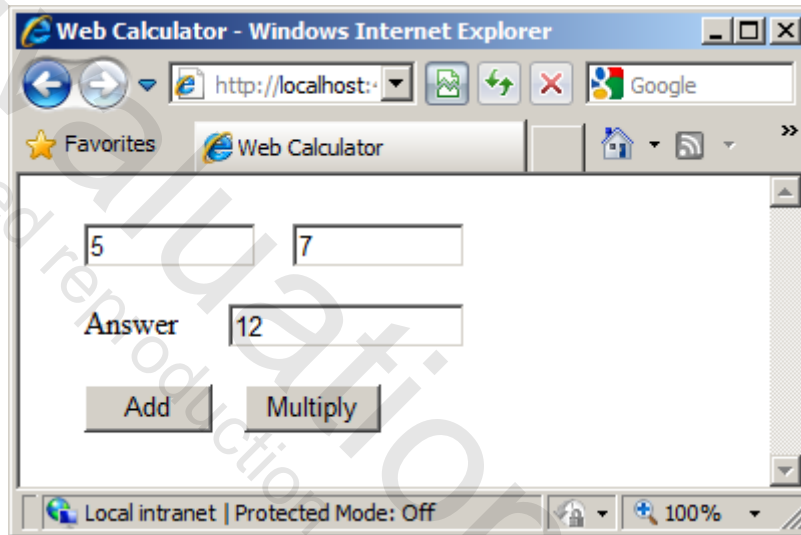
  – An initial Add works fine, but if you try Multiply or adding different numbers, the application behaves strangely.

  – In the demo on the following page, your results may vary depending on how long it takes you to perform the various steps.

Debugging Calculator

1. Examine the **Web.Config** file.

```
<configuration>

    <system.web>
      <compilation debug="true"
                   targetFramework="4.5" />
      <httpRuntime targetFramework="4.5" />
    </system.web>

</configuration>
```

2. Set breakpoints at the first instructions of the handlers for the Add and Multiply buttons.

```
Default.aspx.cs  ⊞ ×
_Default                                    ▼  btnMultiply_Click(object sender, EventArgs e)
        protected void btnAdd_Click(object sender, EventArgs e)
        {
●           int x = Convert.ToInt32(txtX.Text);
            int y = Convert.ToInt32(txtY.Text);
            int z = x + y;
            txtAnswer.Text = z.ToString();
        }
        protected void btnMultiply_Click(object sender, EventArgs e)
        {
●           int x = Convert.ToInt32(txtX.Text);
            int y = Convert.ToInt32(txtY.Text);
            int z = x * y;
            txtAnswer.Text = z.ToString();
        }
    }
```

3. Run under the debugger (Debug | Start or F5 or the toolbar button ▶ Firefox ▾ [1]). If see a dialog box asking if you want to enable debugging, click OK.

---

[1] Whatever is your default browser will be shown as the text in the button, unless you choose a different browser from the dropdown.

Debugging Calculator (Cont'd)

4. Enter some numbers to add, and click the Add button. You should hit a breakpoint.

```
Web.Config        Default.aspx.cs   ⊞ ✕
_Default                                                    ⊙ btnAdd_Click(object sender, EventArgs e)
        protected void btnAdd_Click(object sender, EventArgs e)
        {
●           int x = Convert.ToInt32(txtX.Text);
            int y = Convert.ToInt32(txtY.Text);
            int z = x + y;
            txtAnswer.Text = z.ToString();
        }
        protected void btnMultiply_Click(object sender, EventArgs e)
        {
●           int x = Convert.ToInt32(txtX.Text);
            int y = Convert.ToInt32(txtY.Text);
            int z = x * y;
            txtAnswer.Text = z.ToString();
        }
}
```

5. Continue (F5 or toolbar button ▶ Continue ▾).

6. Click the Multiply button. You don't hit the breakpoint!

7. Try adding different numbers. You may see the original numbers brought back! And no breakpoint!

Application-Level Tracing

- **In this case, debugging has not revealed the problem.**

  – We did not hit breakpoints, so we could not do things like inspect the value of the parameters that came in to the server from the client.

- **Another diagnostic tool is tracing, which can be enabled at both the application level and page level.**

- **You enable tracing at the application level by setting** *trace enabled* **to** *true* **in the** *Web.config* **file.**

  – To avoid clutter in the trace output, also set **debug** to **false**.

  ```
  <compilation debug="false"
                targetFramework="4.5">
  </compilation>
  <trace enabled="true"/>
  ...
  ```

  – With the **pageOutput** setting you can have the output shown directly on the page.

  ```
  <trace enabled="true" pageOutput="true"/>
  ```

Tracing Calculator

1. In the **CalculatorVs** application enable tracing in the **Web.config** file as shown on the preceding page.

```
<trace enabled="true"/>
```

2. To ensure that **Default.aspx** is the first page seen in the application, make that it is your start page. (Right-click over **Default.aspx** and choose Set As Start Page from the context menu.)

3. Run the application, not in the debugger.

4. Enter the numbers 5 and 77 and click the Add button. This will bring up the browser at a URL like **http://localhost:49490/Default.aspx**.
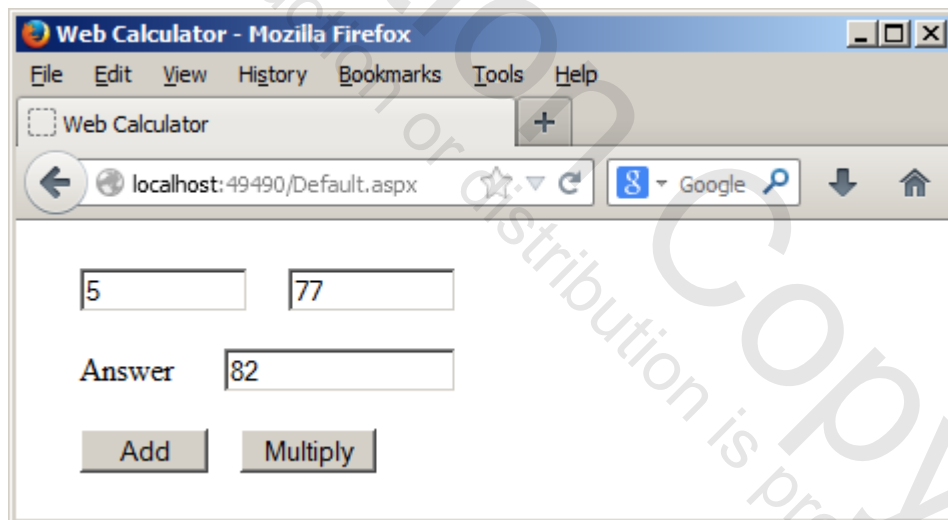
5. Make the second number 777 and click Add again. You should see the results displayed of the first addition and not the second, and the second number has reverted to 77.

Tracing Calculator (Cont'd)

6. To see the trace, navigate in the browser to the **Trace.axd** URL at the root of the application directory. (The URL should look like **http://localhost:49490/Trace.axd.)**

   − If you see other requests in the trace, clear the current trace and try again.
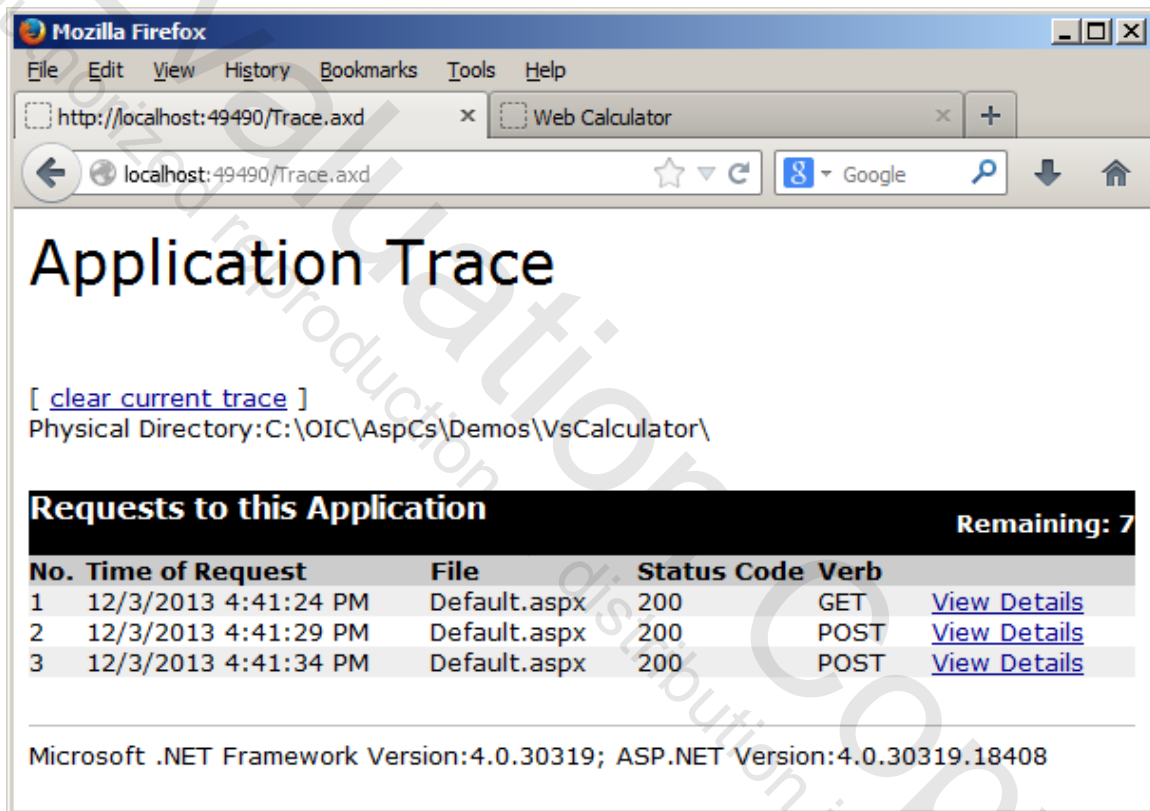
7. There is one GET, from the first view of the page, and two POST, one for each time you clicked the Add button.

Tracing Calculator (Cont'd)

8. View the details of the first POST. Look in the Form collection.

Form Collection	
Name	**Value**
__VIEWSTATE	/wEPDwULLTExMTI5OTk1ODVkZDV7JarEkD8nTKfpYjDF3rUTSCk
__EVENTVALIDATION	/wEWBgKmi9OKCgKShpCYCAKShvy8DwLdsKuIBAKMk4HYDwLsn
txtX	5
txtY	77
txtAnswer	
btnAdd	Add

9. View the details of the second POST.

Form Collection	
Name	**Value**
__VIEWSTATE	/wEPDwULLTExMTI5OTk1ODVkZDV7JarEkD8nTKfpYjDF3rUTSCk
__EVENTVALIDATION	/wEWBgKmi9OKCgKShpCYCAKShvy8DwLdsKuIBAKMk4HYDwLsn
txtX	5
txtY	777
txtAnswer	82
btnAdd	Add

10.   It is clear that the second number, 777, *did* reach the server.

Using the Page Cache

- **By this time you may have realized what the problem likely is: we have cached this page!**

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default" %>
<%@OutputCache Duration="15" VaryByParam="none"%>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Web Calculator</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:TextBox ID="txtX" runat="server"
Style="z-index: 100; left: 32px; position:
absolute; top: 24px" Width="80px">
</asp:TextBox>
        <asp:TextBox ID="txtY" runat="server"
Style="z-index: 101; left: 136px; position:
absolute; top: 24px" Width="80px">
</asp:TextBox>
...

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

- The version of the program that does tracing is saved in **VsCalculator\Step1** in the chapter folder.

Single Page Example

- **Let's look some more at debugging and tracing.**

- **As an illustration, consider the *Default.aspx* page in the *Calculator* folder in this chapter, at the URL:**

http://localhost/AspCs/Chap09/Calculator/Default.aspx

  – The core logic is complete in one page, with no code-behind.

  – This version of the program use flow layout instead of absolute positioning, and there is a Divide button.
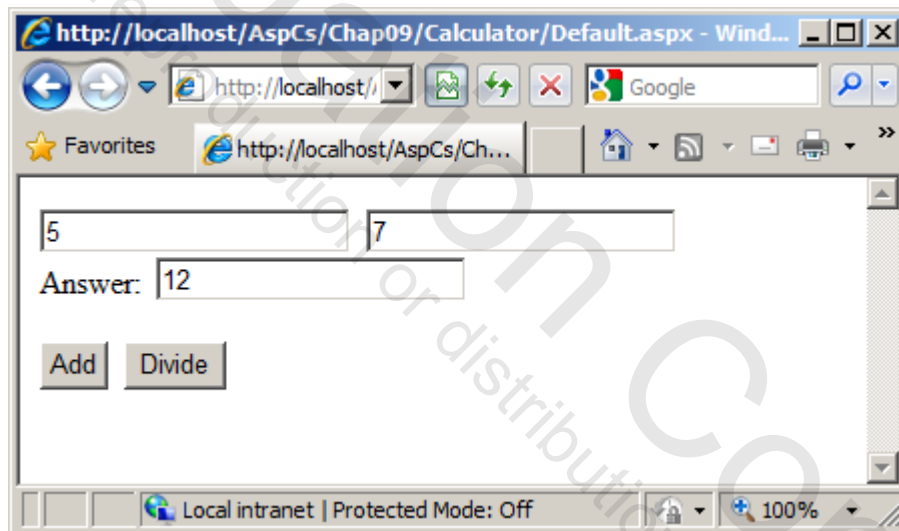
Preparing to Debug

- **You can enable debugging on a particular page by setting the *Debug* attribute of the *Page* directive to *true*.**

```
<%@ Page Language="C#" Trace="false" Debug="true"%>
<html>
<head>
   <title>Calculator</title>
   <script runat="server">
   void cmdAdd_Click(object source,
                     EventArgs e)
   {
      Trace.Write("Add called");
      int x = Convert.ToInt32(txtX.Text);
      int y = Convert.ToInt32(txtY.Text);
      int z = x + y;
      txtAnswer.Text = z.ToString();
   }
   ...
```

- **You can enable debugging for an entire application by setting *debug* to *true* in the compilation element of the *Web.config* file.**

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.web>
   ...
      <compilation defaultLanguage="c#"
                   debug="true"
                   targetFramework="4.5"/>
   ...
   </system.web>
</configuration>
```

Trace Messages

- **Besides the standard trace output, you may write custom trace messages using the *Trace* property of the *Page* class.**

- **As an example, see the Add handler in our *Calculator/Default.aspx* page.**

```
...
<script runat="server">
void cmdAdd_Click(object source,
                  EventArgs e)
{
   Trace.Write("Add called");
   int x = Convert.ToInt32(txtX.Text);
   int y = Convert.ToInt32(txtY.Text);
   int z = x + y;
   txtAnswer.Text = z.ToString();
}
...
```
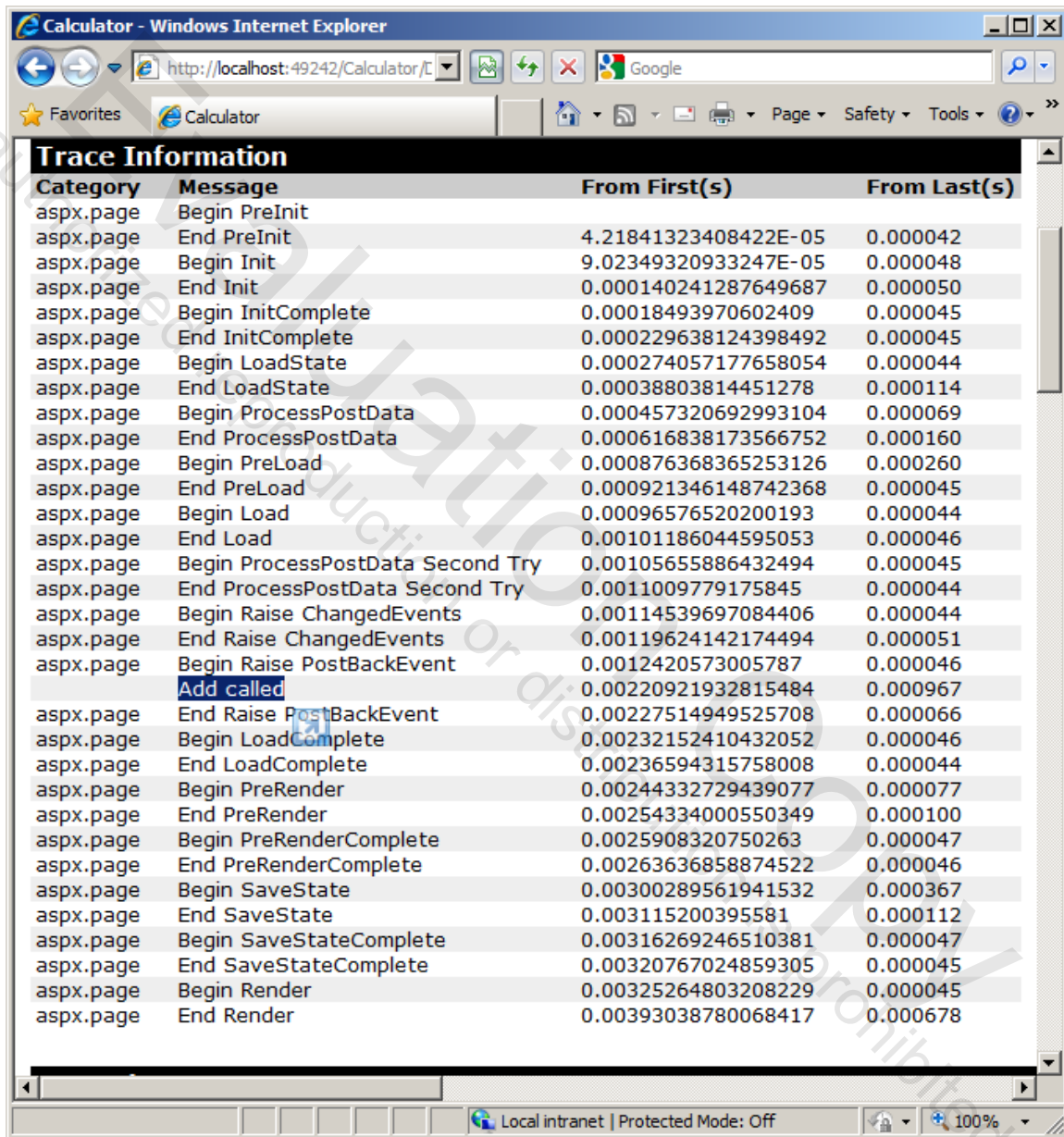
- **To enable tracing on a page, set the *Trace* attribute of the *Page* directive to *true*.**

```
<%@ Page Language="C#" Trace="true" Debug="true" %>
...
```

Tracing the Calculator Page

- ## **We can then see our custom trace message displayed when we click the Add button.**

Category	Message	From First(s)	From Last(s)
	Trace Information		
aspx.page	Begin PreInit		
aspx.page	End PreInit	4.21841323408422E-05	0.000042
aspx.page	Begin Init	9.02349320933247E-05	0.000048
aspx.page	End Init	0.000140241287649687	0.000050
aspx.page	Begin InitComplete	0.00018493970602409	0.000045
aspx.page	End InitComplete	0.000229638124398492	0.000045
aspx.page	Begin LoadState	0.000274057177658054	0.000044
aspx.page	End LoadState	0.00038803814451278	0.000114
aspx.page	Begin ProcessPostData	0.000457320692993104	0.000069
aspx.page	End ProcessPostData	0.000616838173566752	0.000160
aspx.page	Begin PreLoad	0.000876368365253126	0.000260
aspx.page	End PreLoad	0.000921346148742368	0.000045
aspx.page	Begin Load	0.00096576520200193	0.000044
aspx.page	End Load	0.00101186044595053	0.000046
aspx.page	Begin ProcessPostData Second Try	0.00105655886432494	0.000045
aspx.page	End ProcessPostData Second Try	0.0011009779175845	0.000044
aspx.page	Begin Raise ChangedEvents	0.00114539697084406	0.000044
aspx.page	End Raise ChangedEvents	0.00119624142174494	0.000051
aspx.page	Begin Raise PostBackEvent	0.0012420573005787	0.000046
	Add called	0.002220921932815484	0.000967
aspx.page	End Raise PostBackEvent	0.00227514949525708	0.000066
aspx.page	Begin LoadComplete	0.00232152410432052	0.000046
aspx.page	End LoadComplete	0.00236594315758008	0.000044
aspx.page	Begin PreRender	0.00244332729439077	0.000077
aspx.page	End PreRender	0.00254334000550349	0.000100
aspx.page	Begin PreRenderComplete	0.0025908320750263	0.000047
aspx.page	End PreRenderComplete	0.00263636858874522	0.000046
aspx.page	Begin SaveState	0.00300289561941532	0.000367
aspx.page	End SaveState	0.003115200395581	0.000112
aspx.page	Begin SaveStateComplete	0.00316269246510381	0.000047
aspx.page	End SaveStateComplete	0.00320767024859305	0.000045
aspx.page	Begin Render	0.00325264803208229	0.000045
aspx.page	End Render	0.00393038780068417	0.000678

Conditional Tracing

- **You can make the execution of your trace statements conditional on tracing being enabled by testing the *IsEnabled* property of *Trace*.**

```
void cmdDivide_Click(object source,
                     EventArgs e)
{
  if (Trace.IsEnabled)
      Trace.Write("Custom", "Divide called");
  ...
```
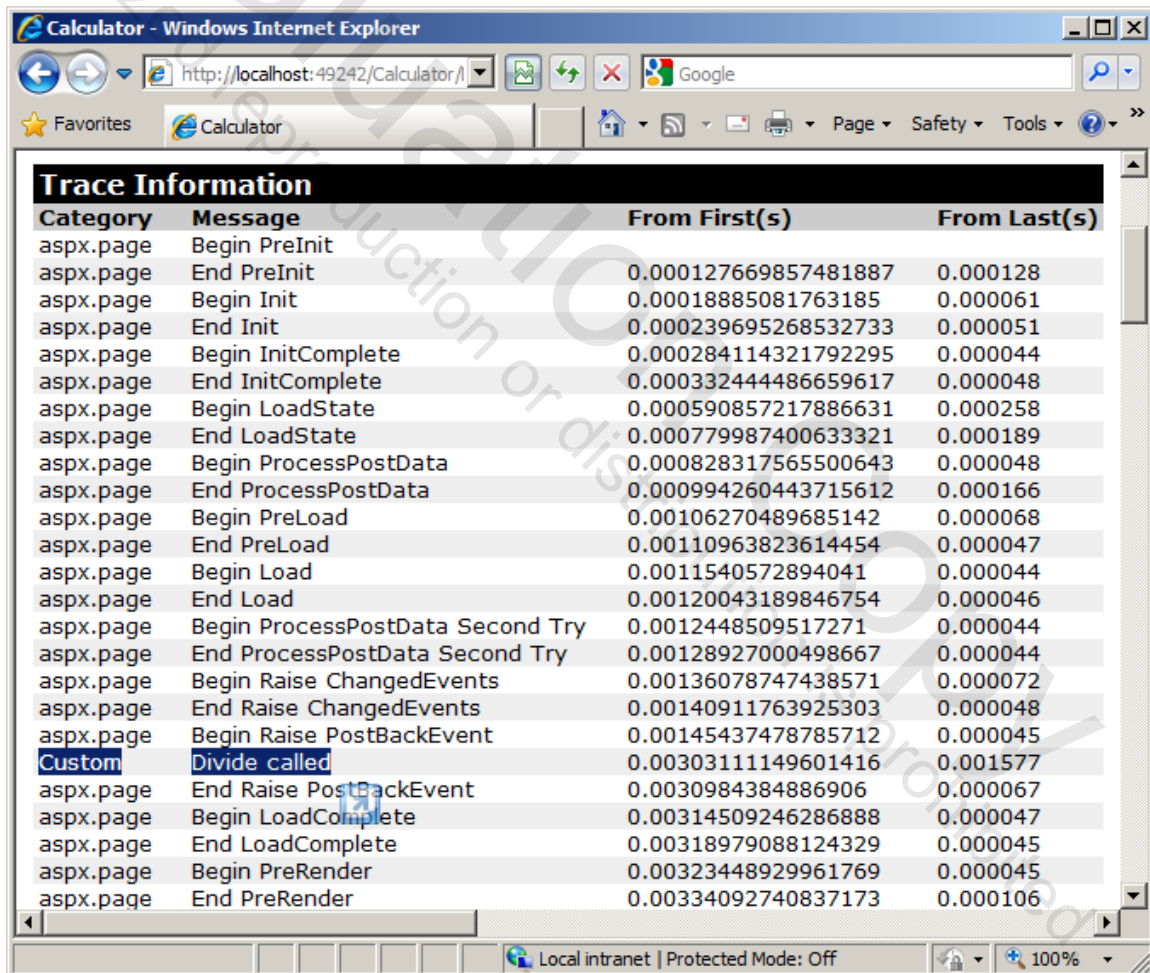
   – You can verify by placing a breakpoint on the **Trace.Write()** statement, with debugging enabled and tracing disabled.

- **What this means is that you can embed useful trace statements in a production application without concern about output or computation time in the normal case when tracing is disabled.**

   – When a problem arises, you can then enable tracing to obtain useful diagnostic information.

Trace Category

- **An overloaded version of the *Trace.Write()* method allows you to specify an optional category for the trace message.**

```
if (Trace.IsEnabled)
   Trace.Write("Custom", "Divide called");
...
```

   – This category will be displayed in the trace output.

Trace Warning

- **The *Warn* method of *Trace* will cause the trace output to be displayed in red.**

  – In our example, we display a warning on an attempt to divide by zero.

```
try
{
   int x = Convert.ToInt32(txtX.Text);
   int y = Convert.ToInt32(txtY.Text);
   if (y == 0)
      Trace.Warn("Custom",
               "Attempting divide by zero");
   ...
```
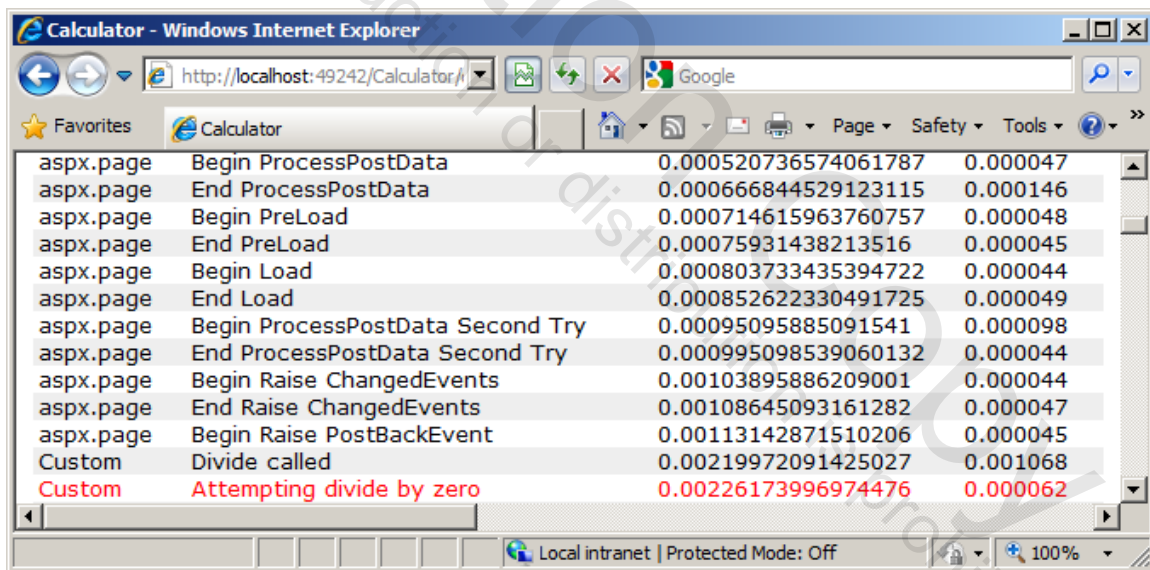
Exceptions in Trace

- ## You may pass an Exception object as a third parameter in a call to *Trace.Write()*.

```
catch (Exception ex)
{
    Trace.Write("Custom", "Exception", ex);
}
```

aspx.page	Begin ProcessPostData	0.000520736574061787	0.000047
aspx.page	End ProcessPostData	0.000666844529123115	0.000146
aspx.page	Begin PreLoad	0.000714615963760757	0.000048
aspx.page	End PreLoad	0.00075931438213516	0.000045
aspx.page	Begin Load	0.000803733435394722	0.000044
aspx.page	End Load	0.000852622330491725	0.000049
aspx.page	Begin ProcessPostData Second Try	0.00095095885091541	0.000098
aspx.page	End ProcessPostData Second Try	0.0009950985390060132	0.000044
aspx.page	Begin Raise ChangedEvents	0.00103895886209001	0.000044
aspx.page	End Raise ChangedEvents	0.00108645093161282	0.000047
aspx.page	Begin Raise PostBackEvent	0.00113142871510206	0.000045
Custom	Divide called	0.00219972091425027	0.001068
Custom	Attempting divide by zero	0.00226173996974476	0.000062
Custom	Exception Attempted to divide by zero. at ASP.default_aspx.cmdDivide_Click (Object source, EventArgs e) in c:\OIC\AspCs\Chap09 \Calculator\Default.aspx:line 26	0.00245310507341017	0.000191
aspx.page	End Raise PostBackEvent	0.00618486427744308	0.003732
aspx.page	Begin LoadComplete	0.00623570872834397	0.000051
aspx.page	End LoadComplete	0.00628180397229257	0.000046

Errors in ASP.NET

- **Another benefit to developing Web applications using ASP.NET is the robust exception handling provided in all .NET languages.**

  – This is in contrast to the weak support for exceptions offered by the scripting languages used with ASP.

- **Our sample *Calculator* page has already illustrated handling exceptions.**

- **But what happens if an uncaught exception is encountered?**

- **To see the result, you can throw an exception in our catch handler.**

```
catch (Exception ex)
{
   Trace.Write("Custom", "Exception", ex);
   throw ex;
}
```

Uncaught Exception

- ## ASP.NET displays a generic error page for errors, including exceptions that are not caught in application code.

  - If we divide by zero in our modified **Calculator** page, we will see this error page:
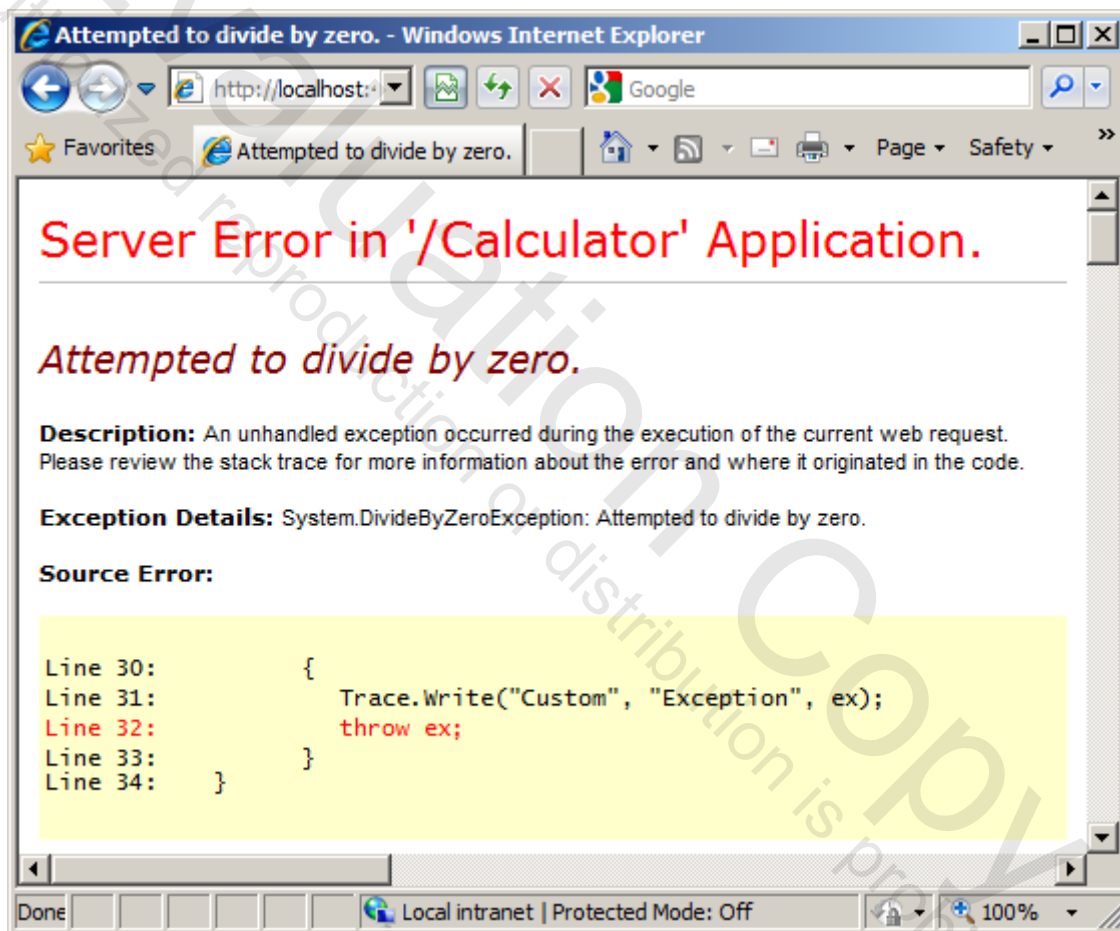
Custom Error Pages

- **Normally you would not want your users to see the generic error page displayed by ASP.NET.**

- **You can create custom error pages, which will be displayed as you direct.**

  – A particular error page can be designated on a page-by-page basis by using the **ErrorPage** attribute of the **Page** directive.

```
<%@ Page Language="C#" Trace="false" Debug="true"
    ErrorPage="PageError.html" %>
```

  – A default error page can be provided for the entire application by using the **customErrors** element of the **Web.config** file.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.web>
      <customErrors
          defaultRedirect='GeneralError.html'
          mode='On'>
      </customErrors>
      ...
   </system.web>
</configuration>
```

Lab 9

Error Pages for Calculator Application

In this lab you will create error pages for the Calculator example. The first error page is a general error page that applies to the entire application, and the second error page is specific to **Calculator.**.

Detailed instructions are contained in the Lab 9 write-up at the end of the chapter.

Suggested time: 30 minutes

Summary

- **You can debug ASP.NET applications created using Visual Studio in the same manner as you would debug other .NET applications.**

- **To perform tracing at the application level, enable tracing in the *Web.config* file.**

- **You can enable tracing at the page level with the *Trace* attribute of the *Page* directive.**

- **.NET exceptions provide a robust exception handling mechanism for ASP.NET application.**

- **You can provide custom error pages to be displayed for uncaught exceptions.**

Lab 9

Error Pages for Calculator Application

Introduction

In this lab you will create error pages for the Calculator example. The first error page is a general error page that applies to the entire application, and the second error page is specific to **Calculator**.

Suggested Time: 30 minutes

Root Directory: OIC\AspCs

Directories: Labs\Lab9\Error   (do your work here)
                   Chap09\Calculator   (contains backup of starter page)
                   Chap09\Error   (answer)

Instructions:

1. Bring up the starter page using Visual Web Developer.

2. Verify that if you try to divide by zero, no exception is thrown (it is caught). If the answer textbox is blank, it will remain blank. Also, verify that you can perform additions and divisions if divisor is not zero.

3. In the **catch** block, add code to rethrow the Exception object. Display the page again in the browser and observe the generic error page. (Notice that if you do not have debugging turned on for the page, the error page will give a message explaining how you can enable debugging.)

4. Create an HTML page **GeneralError.html** that displays a simple message saying there was an error, try again. This could be invoked by accessing a bad file name.

5. In **Web.config** add a **customErrors** element that will redirect to the general error page you created in the previous step.

6. Test in the browser. Observe that this page will be displayed when you try to divide by zero (and also if you have illegal data in one of the input textboxes.)

7. Create an HTML page **PageError.html** that displays a different message.

8. Hook this new error page to **Default.aspx** by using the **ErrorPage** attribute of the **Page** directive.

9. Run again in the browser. When there is an input error, you should now see your new error page displayed in place of the original general error page you created