

# **ASP.NET Using C#**

## *Student Guide*

**Revision 3.1**

# **ASP.NET Using C#**

## **Rev. 3.1**

### **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.

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

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

Copyright ©2008 Object Innovations Enterprises, LLC All rights reserved.

Object Innovations  
877-558-7246  
[www.objectinnovations.com](http://www.objectinnovations.com)

Printed in the United States of America.

# Table of Contents (Overview)

Chapter 1	Introduction to ASP.NET
Chapter 2	Web Forms Architecture
Chapter 3	ASP.NET and HTTP
Chapter 4	Web Applications Using Visual Studio
Chapter 5	State Management and Web Applications
Chapter 6	Server Controls
Chapter 7	Caching in ASP.NET
Chapter 8	ASP.NET Configuration and Security Fundamentals
Chapter 9	Debugging, Diagnostics and Error Handling
Chapter 10	More Server Controls
Chapter 11	ADO.NET and LINQ
Chapter 12	Data Access in ASP.NET 3.5
Chapter 13	Personalization and Security
Chapter 14	Introduction to ASP.NET AJAX
Chapter 15	HTTP Pipeline
Appendix A	Learning Resources
Appendix B	Configuring IIS for ASP.NET 3.5

# 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)

<b>Chapter 1 Introduction to ASP.NET .....</b>	<b>1</b>
Web Application Fundamentals.....	3
Setting up the Web Examples .....	4
Creating a Virtual Directory .....	7
Home Page for ASP.NET Examples .....	9
Benefits of ASP.NET.....	10
ASP.NET Example Program .....	11
An Echo Program.....	12
ASP.NET Features.....	15
Compiled Code .....	17
Server Controls .....	18
Browser Independence.....	19
Separation of Code and Content .....	20
State Management.....	21
Lab 1 .....	22
Summary .....	23
<b>Chapter 2 Web Forms Architecture .....</b>	<b>27</b>
Web Forms Architecture.....	29
Code-Behind Version of Echo Example.....	30
HelloCodebehind.aspx .....	31
HelloCodebehind.aspx.cs.....	32
Page Class .....	33
Inheriting from Page Class .....	34
Web Forms Page Life Cycle.....	36
View State.....	39
Web Forms Event Model .....	40
Page Processing .....	42
Page Events.....	43
Page Properties .....	44
Sample Program.....	45
Page Directive.....	49
Tracing .....	51
Code-Behind in ASP.NET 2.0.....	53
ASP.NET 2.0 Code-Behind Example.....	54
ASP.NET 2.0 Code-Behind Page .....	55
Lab 2 .....	56
Summary .....	57
<b>Chapter 3 ASP.NET and HTTP .....</b>	<b>61</b>
Classical Web Programming.....	63
Active Server Pages Object Model.....	64
Request and Response Objects .....	65

Request/Response in ASP.NET .....	66
HttpRequest Class .....	67
Properties of HttpRequest .....	68
Using HttpRequest Class .....	69
HTTP Collections .....	70
HttpResponse Class .....	72
Redirect .....	73
HttpUtility .....	74
Echo Program Example .....	75
Echo.aspx .....	76
EchoBack.aspx.....	77
GET and POST Compared.....	78
QueryString and Forms Collections .....	79
Lab 3 .....	80
Summary .....	81
<b>Chapter 4 Web Applications Using Visual Studio .....</b>	<b>85</b>
Using Visual Studio .....	87
Visual Web Developer.....	88
Visual Web Developer Demo .....	89
Using Components in ASP.NET 2.0.....	95
Compilation Error .....	96
Shadow Copying.....	100
Shadow Copy Demonstration .....	101
Temporary Copy of the Component .....	104
ASP.NET Applications .....	106
Sessions.....	107
Global.asax .....	108
Web Application Life Cycle .....	109
Application Life Cycle Example .....	110
Global.asax .....	111
Log Class .....	112
Sample Log File.....	113
StringStore Class.....	114
Data Binding .....	115
Data Binding Code Example .....	116
Session Data.....	117
Sessions Using IIS .....	118
Absolute Positioning.....	120
Adding Global.asax File .....	121
XHTML .....	122
XHTML in Visual Studio .....	123
Lab 4 .....	124
Summary .....	125
<b>Chapter 5 State Management and Web Applications .....</b>	<b>133</b>

Session and Application State.....	136
Example Program.....	137
Session Object.....	138
Page_Load.....	139
Session Variable Issues.....	141
Session State and Cookies .....	142
Session State Timeout.....	143
Session State Store.....	144
Application State.....	145
Implementing Application State .....	146
Global.asax .....	147
Users.aspx.cs.....	148
Multithreading Issues.....	149
Bouncing the Web Server .....	150
Lab 5A .....	151
Cookies .....	152
Cookies and ASP.NET.....	153
HttpCookie Properties.....	154
Example – Exposing Cookies .....	155
Acme Travel Agency Case Study .....	161
State Management Techniques .....	163
Lab 5B.....	164
Summary .....	165
<b>Chapter 6 Server Controls .....</b>	<b>179</b>
Server Controls in ASP.NET .....	181
HTML Server Controls .....	182
Using HTML Server Controls .....	183
HTML vs. Web Forms Server Controls.....	184
Server Control Examples .....	185
HTML Controls Example .....	186
Code for Login.....	187
HTML Controls in Visual Studio .....	188
Using HTML Controls.....	189
Web Controls .....	190
Validation Controls.....	191
Required Field Validation.....	193
Regular Expression Validation .....	194
Rich Controls .....	195
Copying a Web Site .....	196
Lab 6 .....	197
User Controls .....	198
Using a User Control .....	199
Copyright.ascx .....	200
Copyright.ascx.cs.....	201
User Control Example.....	202

Summary .....	203
<b>Chapter 7 Caching in ASP.NET .....</b>	<b>207</b>
Introduction.....	209
What is Caching? .....	210
Need for Caching (Why Cache?).....	211
Data to be Cached – Time Frame .....	212
ASP vs. ASP.NET Response Model.....	213
Caching in ASP.NET .....	214
Three Types of Caching in ASP.NET.....	215
Output Caching .....	216
@ OutputCache Directive.....	217
Simple Output Caching Example.....	218
@ OutputCache – Attributes in Detail.....	221
VaryByParam in Detail.....	222
HttpCachePolicy Class .....	224
HttpCachePolicy Class – Example .....	225
Page Fragment Caching.....	227
Common Mistakes in Using Fragment Caching.....	228
Fragment Caching Example.....	229
Data Caching or Application Caching.....	230
Add an Item to the Cache Object.....	231
Insert and Add Methods.....	232
Application Caching – Example .....	233
Expiration.....	235
Problems in Caching.....	236
Lab 7 .....	237
Summary .....	238
<b>Chapter 8 ASP.NET Configuration and Security Fundamentals .....</b>	<b>243</b>
One-minute Introduction to XML! .....	245
ASP.NET Configuration - Overview .....	246
Multi-level Configuration .....	247
Configuration Hierarchy .....	248
Example of Configuration Hierarchy.....	249
Web.Config File Structure .....	250
Web.Config Sections .....	251
Application Settings.....	252
ASP.NET Security – Overview .....	253
Role-Based Security and CAS.....	254
Types and Steps .....	255
Steps in Enabling Role-Based Security .....	256
Three Ways to Authenticate .....	257
Forms Authentication Example .....	258
Forms Authentication – Default.aspx .....	260
Forms Authentication – Web.Config.....	263



Features of Forms Authentication.....	264
Forms Authentication Classes .....	265
Customizing Forms Authentication .....	266
Authentication Source.....	267
Forms Authentication – Analysis .....	268
Windows Authentication .....	269
Windows Authentication – Analysis .....	270
Passport Authentication .....	271
Passport Authentication - Analysis.....	272
Authorization .....	273
Lab 8 .....	274
Summary .....	275
<b>Chapter 9 Debugging, Diagnostics and Error Handling.....</b>	<b>283</b>
ASP.NET Diagnostics.....	285
Debugging Using Visual Studio .....	286
Calculator Example.....	287
Debugging Calculator .....	288
Application-Level Tracing.....	290
Tracing Calculator .....	291
Using the Page Cache .....	294
An ASP.NET Page Without Visual Studio.....	295
Attaching to VS Debugger.....	296
Preparing to Debug .....	298
Trace Messages .....	299
Tracing the Calculator Page.....	300
Conditional Tracing .....	301
Trace Category.....	302
Trace Warning .....	303
Exceptions in Trace .....	304
Errors in ASP.NET .....	305
Uncaught Exception.....	306
Custom Error Pages .....	307
Lab 9 .....	308
Summary .....	309
<b>Chapter 10 More Server Controls.....</b>	<b>311</b>
ASP.NET 2.0 Control Improvements .....	313
New Controls in ASP.NET 2.0 .....	314
Master Pages .....	315
Master Page Demonstration.....	316
Using a Menu Control.....	319
Creating Content Pages.....	321
TreeView Control .....	322
Master Page Application.....	323
Visual Studio 2008 Solutions .....	324

Lab 10 .....	325
Summary .....	326
<b>Chapter 11 ADO.NET and LINQ.....</b>	<b>331</b>
ADO.NET .....	333
ADO.NET Architecture .....	334
.NET Data Providers .....	336
ADO.NET Interfaces .....	337
.NET Namespaces .....	338
Connected Data Access .....	339
AcmePub Database .....	340
Creating a Connection .....	341
Using Server Explorer .....	342
Performing Queries .....	343
ADO.NET with ASP.NET .....	344
Web Client Isolation .....	345
Web Client Database Code .....	346
Using Commands .....	348
Creating a Command Object .....	349
Using a Data Reader .....	350
Data Reader: Code Example .....	351
Use of Session State .....	352
Generic Collections .....	353
Executing Commands .....	354
Parameterized Queries .....	355
Parameterized Query Example .....	356
Lab 11A .....	357
DataSet .....	358
DataSet Architecture .....	359
Why DataSet? .....	360
DataSet Components .....	361
DataAdapter .....	362
DataSet Example Program .....	363
Data Access Class .....	364
Retrieving the Data .....	365
Filling a DataSet .....	366
Accessing a DataSet .....	367
Using a Standalone DataTable .....	368
DataTable Update Example .....	369
Adding a New Row .....	371
Searching and Updating a Row .....	372
Deleting a Row .....	373
Row Versions .....	374
Row State .....	375
Iterating Through DataRows .....	376
Command Builders .....	377

Updating a Database .....	378
Language Integrated Query (LINQ) .....	379
Bridging Objects and Data.....	380
LINQ Demo .....	381
Object Relational Designer .....	382
IntelliSense.....	384
Basic LINQ Query Operators .....	385
Obtaining a Data Source .....	386
LINQ Query Example.....	387
Filtering.....	388
Ordering .....	389
Aggregation .....	390
Obtaining Lists and Arrays .....	391
Deferred Execution .....	392
Modifying a Data Source .....	393
Performing Inserts via LINQ to SQL.....	394
Performing Deletes via LINQ to SQL .....	395
Performing Updates via LINQ to SQL .....	396
Lab 11B.....	397
Summary .....	398
<b>Chapter 12 Data Access in ASP.NET 3.5.....</b>	<b>409</b>
Data Access in ASP.NET 2.0 .....	411
Data Access Demonstration.....	412
Data Entry Demonstration .....	417
SQL Generation Options .....	418
Enable Edit and Delete .....	419
Editing Records.....	420
GridView Control .....	421
DetailsView Control .....	422
Storing the Connection String.....	423
Protecting the Configuration String.....	424
Lab 12A .....	425
FormView Control .....	426
Master/Detail Web Pages.....	427
Data Binding .....	430
Template Editing.....	431
Using XML Data.....	433
Example Program.....	434
XML Data Source Demo .....	435
Multiple-Tier Data Access.....	438
Object Data Source .....	440
Lab 12B.....	443
Data Access in ASP.NET 3.5 .....	444
ListView Demonstration.....	445
Configuring the ListView .....	446

ListView Edit .....	447
DataPager Control.....	448
LinqDataSource Control .....	449
Configuring LinqDataSource.....	450
Summary .....	452
<b>Chapter 13 Personalization and Security .....</b>	<b>461</b>
Themes .....	463
Control Skins .....	464
Sample Skin File .....	465
Applying Themes.....	466
Example of Themes .....	467
Security in ASP.NET 2.0.....	469
ASP.NET Membership .....	470
Login Controls .....	471
Membership Demonstration.....	472
Web Site Administration Tool .....	473
Access Rules .....	475
Login Controls Demo .....	477
Running the Membership Demo .....	478
Lab 13A .....	480
Profile Properties .....	481
Profile Demonstration.....	482
Using ASP.NET Profile Properties.....	483
Lab 13B.....	485
Summary .....	486
<b>Chapter 14 Introduction to ASP.NET AJAX.....</b>	<b>497</b>
Desktop Applications.....	499
Web Applications.....	500
Plug-Ins .....	501
Client-Side Scripting.....	502
JavaScript Example.....	503
Script Code .....	504
JavaScript in ASP.NET.....	505
Dynamic Pages.....	506
Efficient Page Redraws.....	507
AJAX .....	508
Google Maps.....	509
ASP.NET AJAX .....	510
Partial Page Rendering.....	511
Partial Page Rendering Example .....	512
UpdatePanel Control.....	513
AJAX Extensions Controls .....	514
UpdatePanel Demo .....	515
AJAX Client Library .....	517

Using the Client Library .....	518
ScriptManager Control .....	519
Embedded JavaScript Files .....	520
Client Library Namespaces .....	521
Sys.Debug Tracing.....	522
Simple Client Library Example .....	523
Document Object Model .....	524
JavaScript for Simple Calculator .....	525
Using the Client Library .....	526
JavaScript in Assemblies .....	527
Providing a ScriptReference .....	528
AJAX Control Toolkit .....	529
ACT Controls in Visual Studio.....	530
ACT Sample Web Site.....	531
AjaxControlToolkit.dll .....	532
Registering AjaxControlToolkit.dll .....	533
Extender Controls .....	534
NumericUpDownExtender Control .....	535
Lab 14 .....	536
Summary .....	537
<b>Chapter 15 HTTP Pipeline.....</b>	<b>543</b>
Web Applications.....	545
ASP.NET Request Processing .....	546
ASP.NET Architecture with IIS 5.0 .....	547
Pipeline Processing .....	548
Pipeline Architecture .....	549
Customizing the HTTP Pipeline .....	550
Customizing Applications.....	551
Customizing a Welcome Application .....	552
Logger Class .....	557
Custom Handlers.....	559
IHttpHandler Interface.....	560
Custom Handler Example .....	561
Custom Handler Configuration.....	563
Custom Handler Demonstration .....	564
Entry in Configuration File.....	565
Extension Mapping in IIS .....	566
.ashx Files .....	570
string.ashx .....	571
Custom Modules .....	572
Example: DemoModule .....	573
Using DemoModule.....	574
Lab 15 .....	575
Summary .....	576

<b>Appendix A Learning Resources .....</b>	<b>581</b>
<b>Appendix B Configuring IIS for ASP.NET 3.5 .....</b>	<b>585</b>
If ASP.NET 2.0 Is Installed .....	586
ASP.NET Versions Side-by-Side .....	587
Configuring for ASP.NET 2.0 .....	588
Installing ASP.NET .....	592

# **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 two programming models provided by ASP.NET, Web Forms and Web services.**
- **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

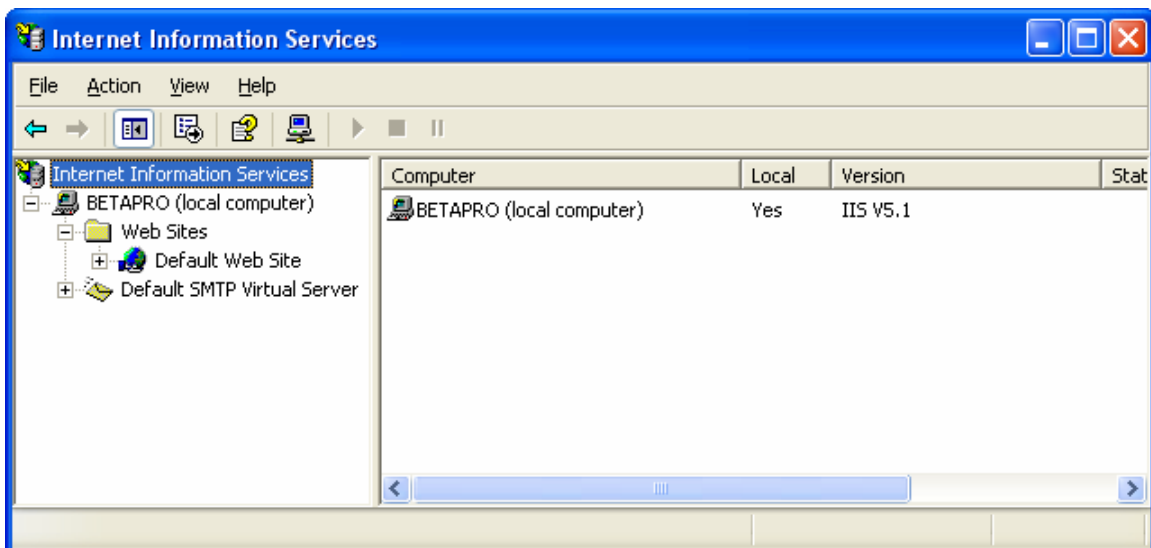
---

- **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, including the sample ASP page, is to have Internet Information Services (IIS) installed on your system.**
  - You will have to explicitly install it with Windows XP.
  - Once installed, you can access the documentation on IIS through Internet Explorer via the URL **`http://localhost`**, which will redirect you to the starting IIS documentation page.
- **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 also run the ASP.NET examples using the built-in ASP.NET Development Server.**
  - This built-in Web server is started automatically from within Visual Studio 2008 and is discussed in Chapter 4.

## Web Examples Setup (Cont'd)

---

- **The figure shows the main window of the Internet Services Manager.**
  - You can Start and Stop the Web server and perform other tasks by right-clicking on Default Web Site.
  - Choosing Properties from the context menu will let you perform a number of configurations on the Web server.



- **The home directory for the default web site is *|Inetpub|wwwroot* on the drive where Windows is installed.**
  - You can change this home directory using Internet Services Manager.

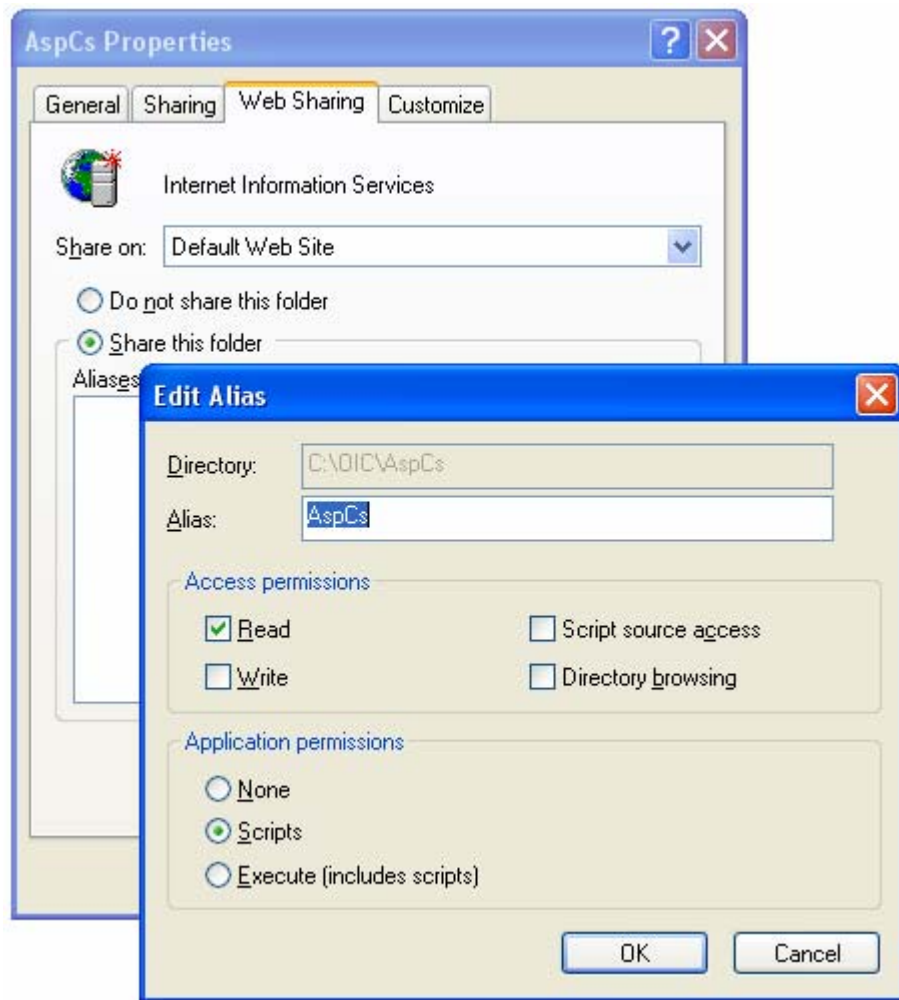
## Web Examples Setup (Cont'd)

---

- **You can access Web pages stored at any location on your hard drive by creating a “virtual directory.”**
  - An easy way to create one is from Windows Explorer. Right-click over the desired directory, choose Properties ..., select the Web Sharing tab, click on the Add button, and enter the desired alias, which will be the name of the virtual directory.
- **The figure on the next page illustrates creating an alias *AspCs*, or virtual directory, for the folder *|OIC\AspCs*.**
  - You should perform this operation now on your own system in order that you may follow along as the course examples are discussed.

# Creating a Virtual Directory

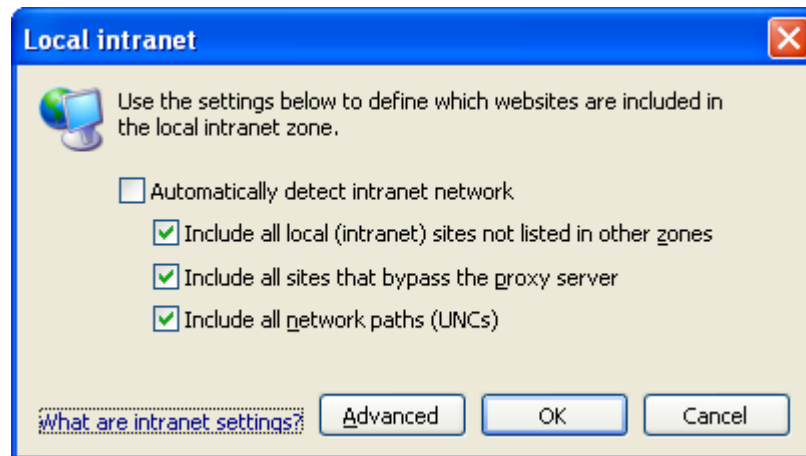
---



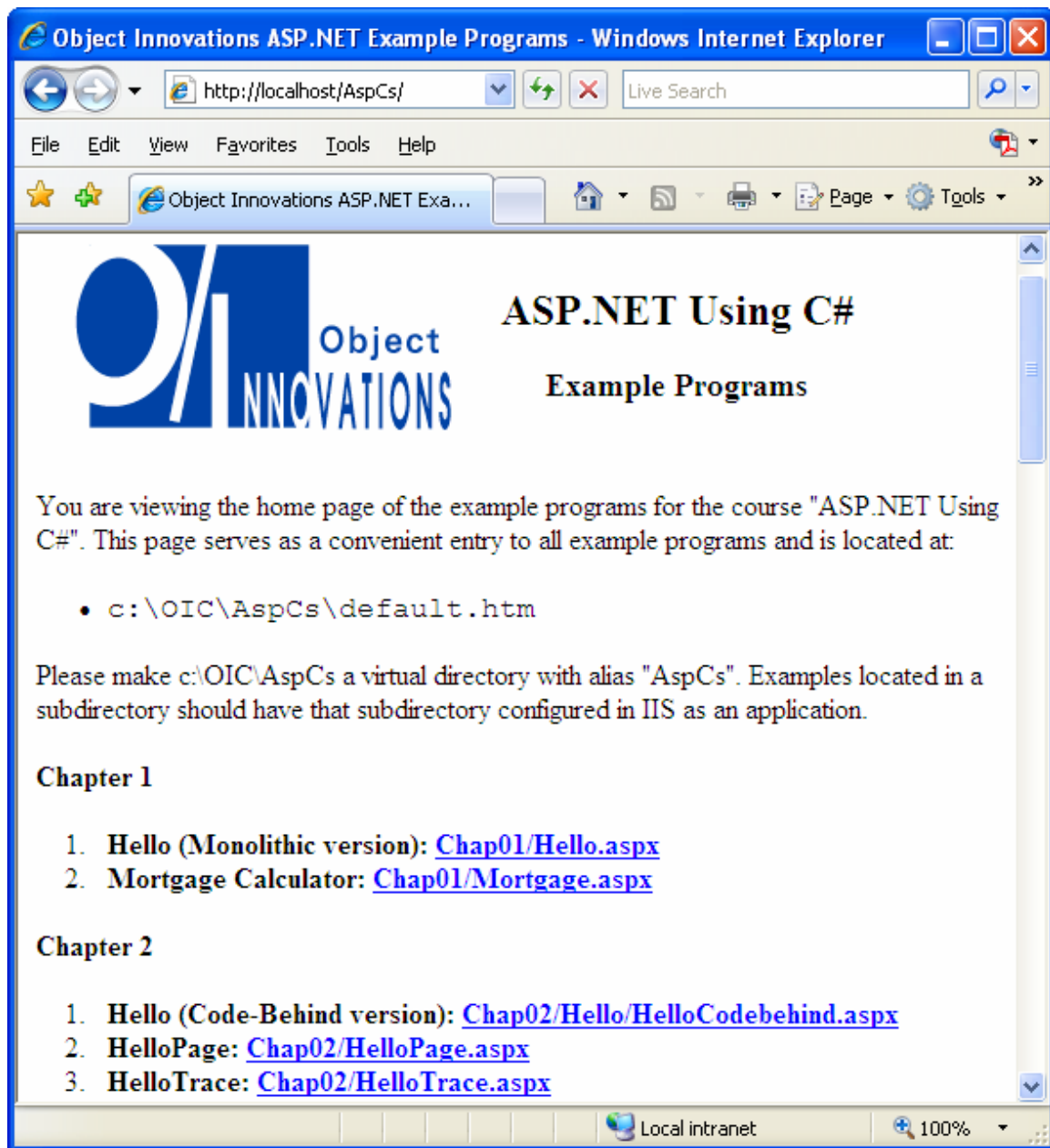
## Web Examples Setup (Cont'd)

---

- **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 this chapter. See the figure on the next page.
- **If you are using Internet Explorer 7 as your Web browser, you should enable local intranet security settings.**
  - From Tools menu select Internet options
  - On Security tab select Local Intranet zone and click the Sites button. Uncheck “Automatically detect ...” and check the others.



# Home Page for ASP.NET Examples



# 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.**
  - A special set of Mobile Controls can emit either HTML or WML, depending on the characteristics of the client device.
- **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 a **.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>
    <SCRIPT RUNAT="SERVER">
        void cmdEcho_Click(object source, EventArgs e)
        {
            lblGreeting.Text="Hello, " + txtName.Text;
        }
    </SCRIPT>
</HEAD>
<BODY>
<FORM RUNAT="SERVER">Your name:&nbsp;
<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>
<P></P>
</FORM>
</BODY>
</HTML>
```

# An Echo Program

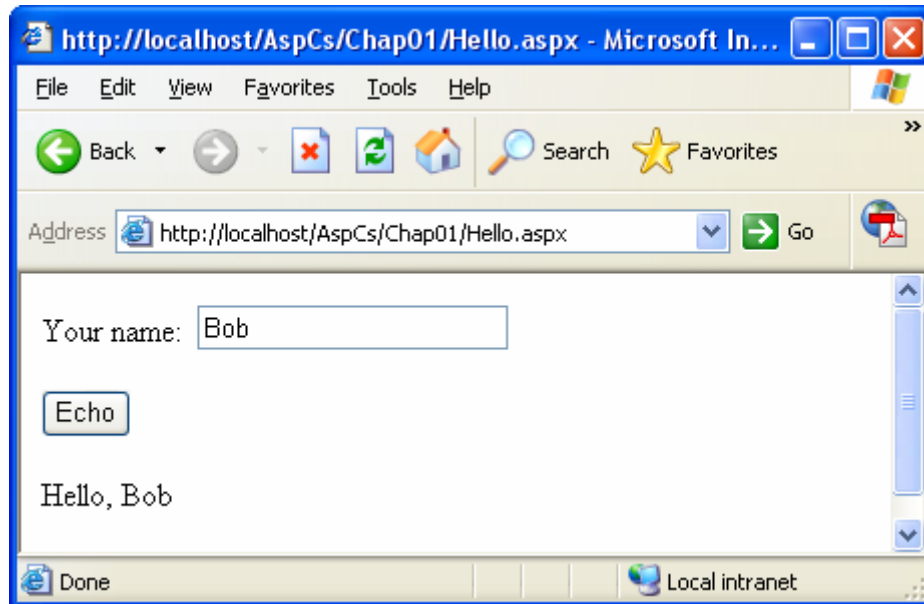
---

- **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 some browsers, such as Internet Explorer, but not others.
- **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.**
- **Two main 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.
- **Web Services 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.
- **This course is focused on Web Forms.**
  - For Web Services the newer technology, Windows Communication Foundation, is preferred.

## ASP.NET Features (Cont'd)

---

- **Both Web Forms and Web Services can 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 runs on top of the common language runtime, including C#, Visual Basic, JScript.NET, and Visual C++.**
  - 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 2.0 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

---

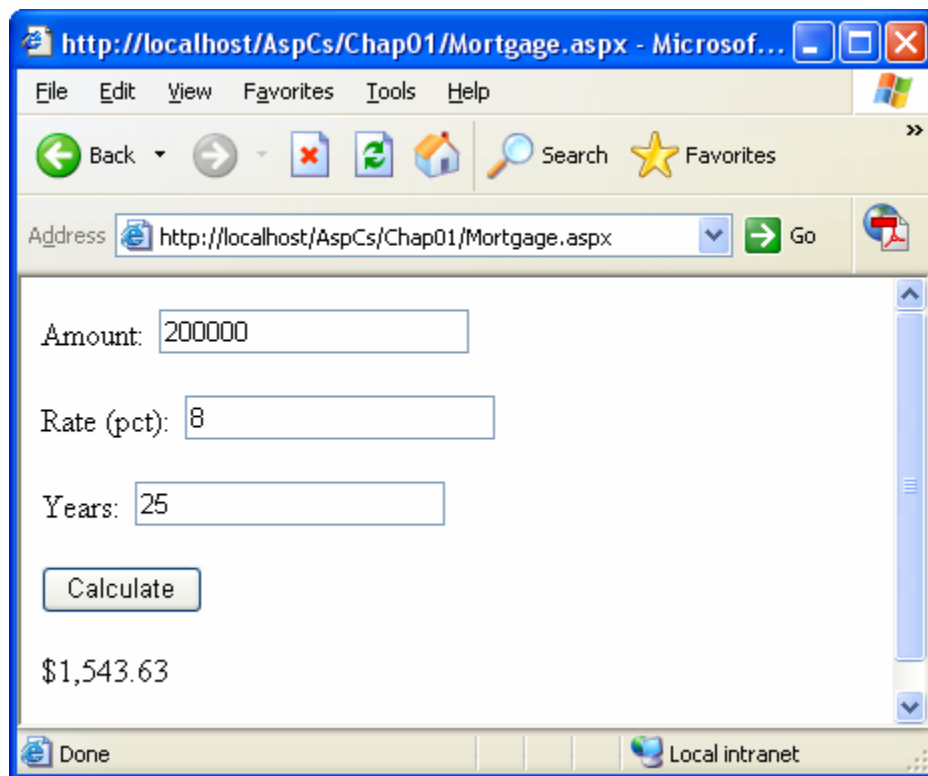
- **HTTP is a stateless protocol.**
- **Thus, if a user enters information in various controls on a form, and sends this filled-out form to the server, the information will be lost if the form is displayed again, unless the Web application provides special code to preserve this state.**
  - ASP.NET makes 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 two programming models, Web Forms and Web Services.**
- **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

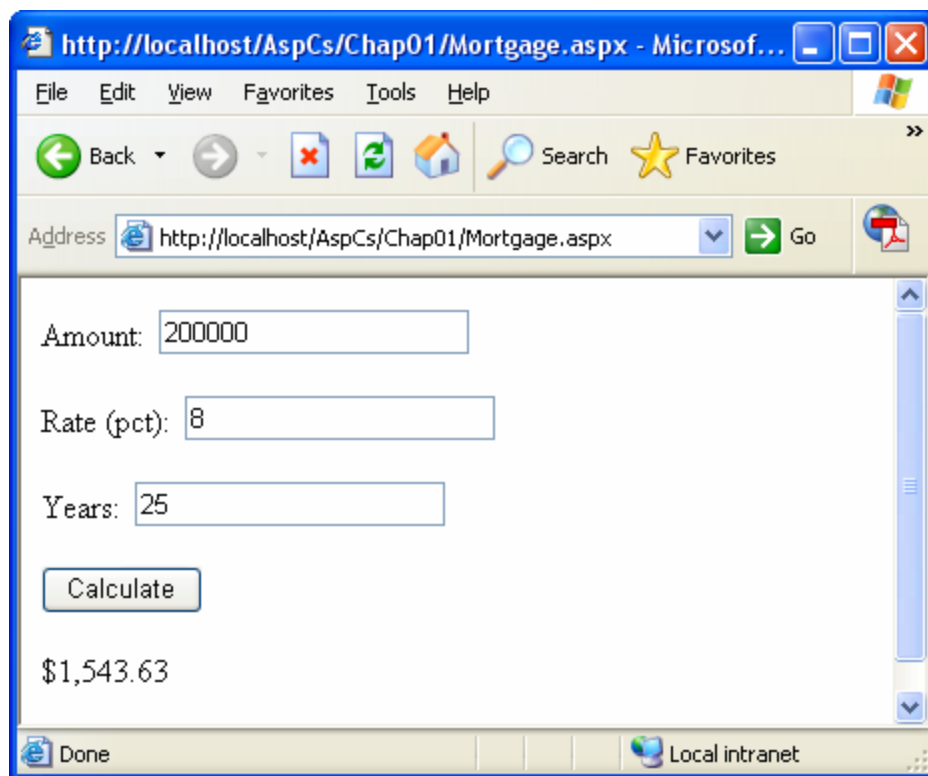
## Lab 1

### 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

<b>Directories:</b>	<b>Labs\Lab1</b>	(do your work here)
	<b>Chap01\Hello.aspx</b>	(starter file)
	<b>Chap01\MortgageCalculator</b>	(C# mortgage calculator)

<b>Files:</b>	<b>Chap01\Mortgage.aspx</b>	(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. 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.
3. Examine the code for the C# console program in the **MortgageCalculator** folder. Run this program a couple of times to generate some test data for use in testing your Web mortgage calculator.
4. 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);
```

5. 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);
```

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





# **Chapter 7**

## **Caching in ASP.NET**

# Caching in ASP.NET

## Objectives

---

*After completing this unit you will be able to:*

- **Answer the questions “What is caching?” and “Why do we need caching?”**
- **Illustrate the three types of caching techniques in ASP.NET.**
- **Explain the concepts of output caching, fragment caching, and application caching in a detailed manner.**
- **Describe the different problems faced while using caching and show their solution.**
- **Outline the principle features of caching and its optimization in ASP.NET.**

# Introduction

---

- **Performance tuning is an exceedingly significant issue to both the programmer writing the code and the system administrator maintaining the application.**
  - In the contemporary high-speed Internet world, performance tuning is a foremost way to enhance the Web applications.
  - The slow Web applications not only waste the company resources, but also drive prospective customers away from the company.
  - Caching is one of the techniques normally used for better performance tuning. Caching will help you to develop ASP.NET Web applications with fast response and suitable levels of throughput.

# What is Caching?

---

- **Caching is a technique used to store some resources or data which are frequently requested by the user, reusing them when needed, so the server does not have to recreate them, thus improving the overall performance.**
  - Retrieving data from memory is much more efficient than from locations such as a database.
  - Thus we store the frequently used data, which are costly to generate, in memory.
  - This technique is widely used to increase performance by keeping frequently accessed or expensive data in memory.
  - Once a page or data has been compiled and served, it is stored in memory, and for the succeeding requests it is satisfied from the cache rather than reinitiating the whole process.

## Need for Caching (Why Cache?)

---

- **In classic ASP, one of the methods developers usually relied on to speed up processing was the use of caching.**
- **By using caching in Web applications, one can significantly enhance performance.**
  - In Web applications, caching is used to hold data across HTTP requests and use them again without the cost of recreating them.
  - Using caching one can avoid a lot of work associated with the retrieving the data from the server (executing ASP.NET code for the request, communicating with server, and getting back the compiled executed query back from the server).
  - The main advantage of caching is based on the data we store. That is, we have to store the data that are frequently requested.

## Data to be Cached – Time Frame

---

- **For what time frame should we cache the data? The answer for this question is based on data (dynamic or static).**
  - We have to cache the data either for an indefinite period or some relative time frame based on either static data or fairly static data used in your application.
  - We have to rebuild the cached content according to the dynamic changes in the database.
  - In ASP page's life cycle the operation such as accessing database information was expensive one. In that scenario we can cache the database information, if the database information is static.

## ASP vs. ASP.NET Response Model

---

- In ASP, the *Response* object used the *Expires*, *ExpiresAbsolute*, and *CacheControl* properties to maintain caching.
- In ASP, the *Response.Expires* property is used to cache a page for a specified number of minutes such as

```
<%Response.Expires = 10%>
```

- The above code sets the cache to expire 10 minutes or 600 seconds from the first request of the page.

- In ASP.NET, the same can be made by as follows:

```
<%Response.Cache.SetExpires(  
    DateTime.Now.AddSeconds( 600 ) )%>
```

- An advantage is that the **DateTime** object we used above in the ASP.NET code provides a remarkable quantity of flexibility in the computation of date and time values.

- One can also use the following *OutputCache* page directive to perform the same operation.

```
<%@ OutputCache Duration = "600" %>
```

- Let us see all those things in detail.

# Caching in ASP.NET

---

- **While caching in classic ASP was a bit of an unpleasant task, it is simple in ASP.NET.**
- **ASP.NET provides simple mechanisms for caching page output or data when they do not need to be computed dynamically for every page request.**
  - There are a number of classes in the .NET Framework designed to aid with caching information.
  - ASP.NET includes a variety of features and tools that allow you to design and implement high-performance Web applications.
  - ASP.NET is the option when you want to create Web applications that will meet the demands of a Web site with high traffic.



# Three Types of Caching in ASP.NET

---

- **ASP.NET provides three types of caching for Web-based applications:**
  - Page Level Caching or Output Caching
  - Page Fragment Caching or Partial-Page Output Caching
  - Programmatic or Data Caching or Application Caching
- **These three types of caching will help you to create high-performance and scalable Web applications.**
  - Output caching allows you to store an entire page.
  - Fragment caching caches portions of a response generated by a request.
  - You can use application caching to programmatically store arbitrary objects to server memory, so that your application can save the time and resources it takes to recreate them.
- **With application caching there are the same multithreading issues that we discussed in connection with application state earlier in the course.**

# Output Caching

---

- **Output caching is a feature of ASP.NET that allows for the contents of an entire page to be stored in the cache.**
  - The cached HTML is sent as a response for the requested ASP.NET page, if the engine finds the requested ASP.NET page in the cached output entry.
  - If it doesn't, then the page is dynamically recreated and the output is stored in the output cache engine.
  - The advantage of output caching is that rather than dynamically executing the ASP.NET page on each request, we serve it statically from memory.
  - This will provide a substantial performance enhancement.

## @ OutputCache Directive

---

- **To be placed in the output cache, a given page response must have public cache visibility and a valid expiration or validation policy.**
  - You can set these using the **@ OutputCache** directive, setting expiration policies using this directive's **Duration** attribute.
  - The following directive activates output caching on the response:

```
<%@ OutputCache Duration="#inseconds"  
VaryByParam="Parametername" %>
```

- **Example:**

```
<%@ OutputCache Duration="20" VaryByParam="none" %>
```

- This directive, added to the top of an ASP.NET page, simply instructs ASP.NET to cache the results of the page for 20 seconds. After 20 seconds the page will re-execute.

# Simple Output Caching Example

---

- **The following example displays the time when the user requests the page.**
  - See **SimpleCache1** in the chapter directory.
- **The example is complete in one file and contains embedded server code.**
  - There is also a configuration file that specifies use of .NET Framework 3.5.

```
<%@ OutputCache Duration="20" VaryByParam="none" %>
```

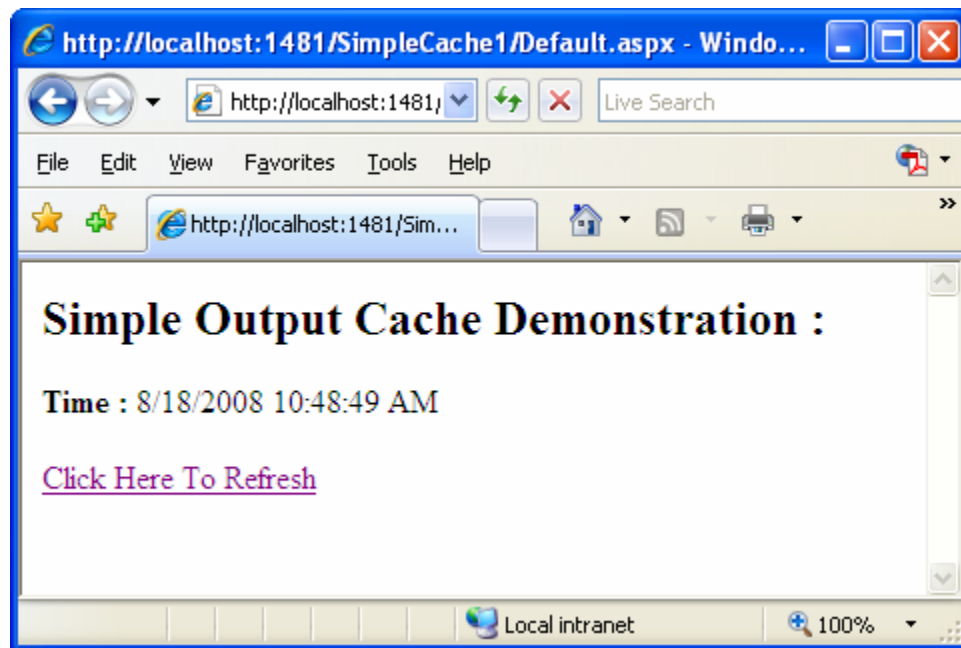
```
<Html>
  <script language="C#" runat="server">
    void Page_Load(Object Src, EventArgs Eve)
    {
      Labell1.Text = DateTime.Now.ToString();
    }
  </script>
  <Body>
    <h2> Simple Output Cache Demonstration :</h2>
    <p><b>Time :</b> <asp:label id="Labell1"
runat="server"/>

    <FORM RUNAT="server">
    <A HREF="Default.aspx">Click Here To Refresh </A>
  </Form>
  </body>
</html>
```

## Output Caching Example (Cont'd)

---

- The figure illustrates a run of this example (using the ASP.NET Developer Server).

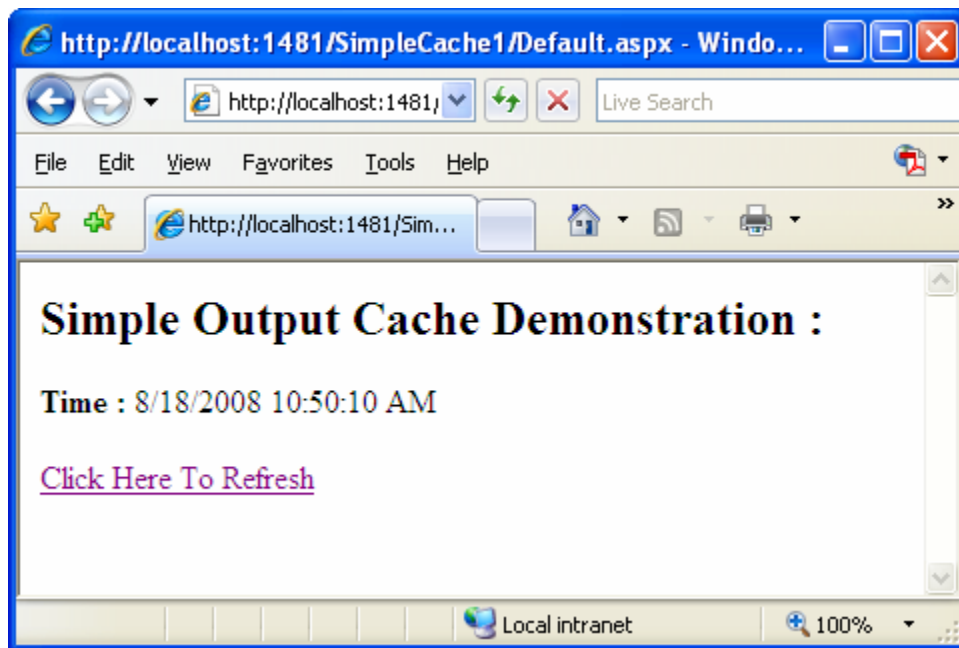


- Note the time at the first time request and you will give notice that the time has not changed after refreshing the page, specifying that the second response be being served from the output cache.
  - The page shows the exact time at the first request.
  - Clicking the refresh button you will notify that the page is served from the cache again.
  - In our example the time duration is specified as 20 seconds.

## Output Caching Example (Cont'd)

---

- **Try again after an elapsed time of 20 seconds or more.**
  - The page in the output cache expires because we specify the time duration only for 20 seconds.



- If you refresh the page the same time is displayed for 20 seconds and after those 20 seconds it displays a later time.
- So from the above program you will see that the page will be served from the output cache for the specified 20 seconds duration.

## @ OutputCache – Attributes in Detail

---

```
<%@ OutputCache Duration="#inseconds"  
VaryByParam="Parametername" %>
```

- The *Duration* parameter denotes the time, in seconds, during which the HTML output of the Web page or user control is cached. The cache becomes invalid if the duration expires.
  - The page is dynamically recreated and the cache repopulated with the newer HTML in the next request.
- The *VaryByParam* parameter is employed to cache the multiple versions of a dynamic page whose content is generated by GET or POST values.
  - If the output used to produce the page is dissimilar for different values passed in via either a GET or POST, this parameter is very useful.
  - This attribute helps us to control how many cached versions of the page should be created based on name/value pairs sent through HTTP POST/GET.
  - The default value is “None”, and if we set the value to “None”, it does not take HTTP GET/POST parameters into account and it caches only the first page.
  - Set the attribute to “\*”, to vary the output cache by all parameter values.

## VaryByParam in Detail

---

- **Let's say you're building a Web application in which a user can view a list of books by a specified publication.**
  - It has one form named **Books.aspx**. The **Books.aspx** provides a list of books.
  - When the user chooses an author and clicks the select button and the updated value is sent back to **Books.aspx**.
- **By default HTTP POST is used.**
  - The data is not appended to the query string as in HTTP GET, but instead name/value pairs is written to a separate line sent with the HTTP header, and the data is not directly visible to the outside world.
- **For simplicity, assume we are using HTTP GET to send data. When you submit data using the GET action, you see as**

`books.aspx?Books= C#, books.aspx?Publication=PHPTR.`

- **If you use “none” you may come across some problems. For example if anyone request with Query string *books.aspx?Books=C#* then after displaying the results, the page is cached.**
- **If another request has a different query string *books.aspx?Books= VB.NET* , the output caching engine will assume that the requests to the two pages are same and return the previous cached page C# instead of VB.NET.**



## VaryByParam in Detail (Cont'd)

---

- So the solution to this problem is you have to specify the *VaryByParam* as *VaryByParam = "Books"*, or by saying to vary on all GET/POST parameters, like: *VaryByParam = "\*" .*
- If we add the following directive to *books.aspx*, then multiple versions of the page (*books.aspx*, *books.aspx?Books= C#*, *books.aspx?Books= VB.NET*, etc..) is cached independently for one hour.

```
<%@ OutputCache Duration="3600" VaryByParam="Books" %>
```

- To output cache pages both based on the Books and Publisher parameter, we would change the directive to:

```
<%@ OutputCache Duration="3600" VaryByParam="Books;Publisher" %>
```

- It is suggested that you not use \*, as this can potentially fill the output cache with pages that aren't frequently accessed. The more specific you make the *VaryByParam*, the more frequently you can serve from the cache.

# HttpCachePolicy Class

---

- The *HttpCachePolicy* class is used to set expirations for a cached page.
  - In the page's code declaration block, or in the code-behind class for the page, include code from the page that sets the expiration policy for the page using **Response.Cache** syntax.
  - The following example sets expirations for the page as the **@ OutputCache** directive does in the previous procedure.

```
Response.Cache.SetExpires(  
    DateTime.Now.AddSeconds(50));  
Response.Cache.SetCacheability(  
    HttpCacheability.Public);
```

- Once the duration for the cached page expires, the next subsequent request for the page causes a dynamically generated response.
  - This response page is cached again for the specified duration.

## HttpCachePolicy Class – Example

---

- Let us look at an example exemplifying programmatic technique for caching, rather than using the *OutputCache* Page directive.
- The following example displays the time similar to the program *SimpleCache1/Default.aspx*:
  - See **SimpleCache2** in the chapter directory.
  - The output is identical for both the programs **SimpleCache1** and **SimpleCache2**.
  - The only difference is that we use **Response.Cache** syntax instead of **@ OutputCache** directive.
- There are two Expiration policies named **Absolute Expiration** and **Sliding Expiration**.
  - In the Absolute Expiration, one can set an absolute or specified time to expire a page.
  - To expire a page from the last time it was accessed, one can use the Sliding Expiration.
  - To set Absolute Expiration the code is

```
Response.Cache.SetSlidingExpiration(false);
```

- To set Sliding Expiration the code is

```
Response.Cache.SetSlidingExpiration(true);
```

## HttpCachePolicy Example (Cont'd)

---

- Here is the example code for *Default.aspx* in *SimpleCache2*.

```
<%@ Page Language="C#" %>
<html>
  <script language="C#" runat="server">
    void Page_Load(Object Src, EventArgs Eve)
    {
      Response.Cache.SetExpires(
        DateTime.Now.AddSeconds(20));
      Response.Cache.SetCacheability(
        HttpCacheability.Public);
      Label1.Text = DateTime.Now.ToString();
    }
  </script>
  <body>
    <h2> Simple Output Cache Demonstration :</h2>
    <p><b>Time :</b> <asp:label id="Label1"
runat="server"/>
    </p>
    <form RUNAT="server">
    <A HREF="Default.aspx">Click Here To Refresh </A>
    </form>
  </body>
</html>
```

# Page Fragment Caching

---

- **Partial-Page Output Caching, or page fragment caching, allows particular portions of pages to be cached.**
- **You have to select particular region of page such that it involve fewer server resources to be generated dynamically for each request.**
  - In other words you have to select the portions of a page that is fairly static.
- **When Fragment Caching?**
  - Sometimes caching only a portion of an entire page is more efficient than caching an entire page.
  - In this case, you have to recognize the objects or data related with the page request that need significant server resources to construct and you can separate them from the rest of the page and cache them for a period of time.
  - If we place an **OutputCache** directive at the top of a user control, the content inside the user control will now be cached for the specified period.
  - For example, if you place the following directive at the top of a user control file, a version of the control is stored in the output cache for 30 seconds.

```
<%@ OutputCache Duration="30" VaryByParam="none" %>
```

# Common Mistakes in Using Fragment Caching

---

- **A user control that is output cached is only dynamically generated for the first request.**
  - Any succeeding requests are fulfilled from the output cache until the control run out.
- **Errors will be generated by code that manipulates a user control containing an @ *OutputCache* directive.**
- **Any programmatic manipulation that must occur to create the content of the user control must be included in the control.**
  - You can include this logic in one of the page lifecycle events associated with the user control, such as in the **Page\_Load** event or **Page\_PreRender** event.

# Fragment Caching Example

---

- The *Copyright* user control example from Chapter 6 includes a count of courses that is read from a flat file.
  - Since the number of courses rarely changes, this is a prime candidate for caching.
  - The program is in the **FragmentCache** folder in this chapter.
  - OutputCache Duration is set to 30 seconds.

```
<%@ Control Language="C#" ...  
<%@ OutputCache Duration="30" VaryByParam="none" %>
```



- You may edit the file *c:\OIC\Data\courses.txt* and observe that the cached data does not change until expiration of the duration interval.
  - Observe what happens when you navigate between the two pages.

# Data Caching or Application Caching

---

- **ASP.NET provides you a solid, user-friendly caching mechanism that allows you to store expensive objects in memory across HTTP requests.**
- **In classic ASP, you store objects in Application scope. Likewise, in ASP.NET you can store objects into a cache using the *Cache* class.**
- **The lifetime of the object is tied to that of the application. When the application is restarted, the instance of its *Cache* object is recreated.**
- **Data caching make use of the .NET Runtime cache engine to store any data or object between responses.**



# Add an Item to the Cache Object

---

- **There are three different techniques to add an item to the Cache object.**
- **Add an item to the cache specifying its key and value:**
  - You can add items to the cache as you would add items to a dictionary, by specifying the item's key and value.
  - The following code adds the item in the cache

```
Cache ["Value1"] = tValue;
```

- To retrieve the data:

```
tValue = Cache["Value1"];  
if (tValue != null)  
    DisplayData(Value1);
```

- **The other two techniques, *Cache.Insert()* method or the *Cache.Add()* method are also used to add items to the Cache.**
  - The **Add()** and the **Insert()** methods operate exactly the same except the **Add()** method returns a reference to the object being inserted to the cache.
  - These methods have the same signatures.
  - To take benefit of the scavenging, expiration, and dependency support offered by the cache, you must use either of the above two methods.

# Insert and Add Methods

---

- ***Cache.Insert()* method**

- The **Insert()** method is overloaded, allowing you to define values for the parameters of the version that you are using.
- For example, to add only an item key and value use the following code.

```
Cache.Insert("Value1", connectionString);
```

- ***Cache.Add()* method**

- The **Add()** method has the same signature as the **Insert()** method, but returns an object representing the item you added.

```
item = Cache.Add("Value1", connectionString);
```

- ***Cache.Remove()* method**

- In some situations you have to remove an item from Cache. In that scenario you can use the following code.

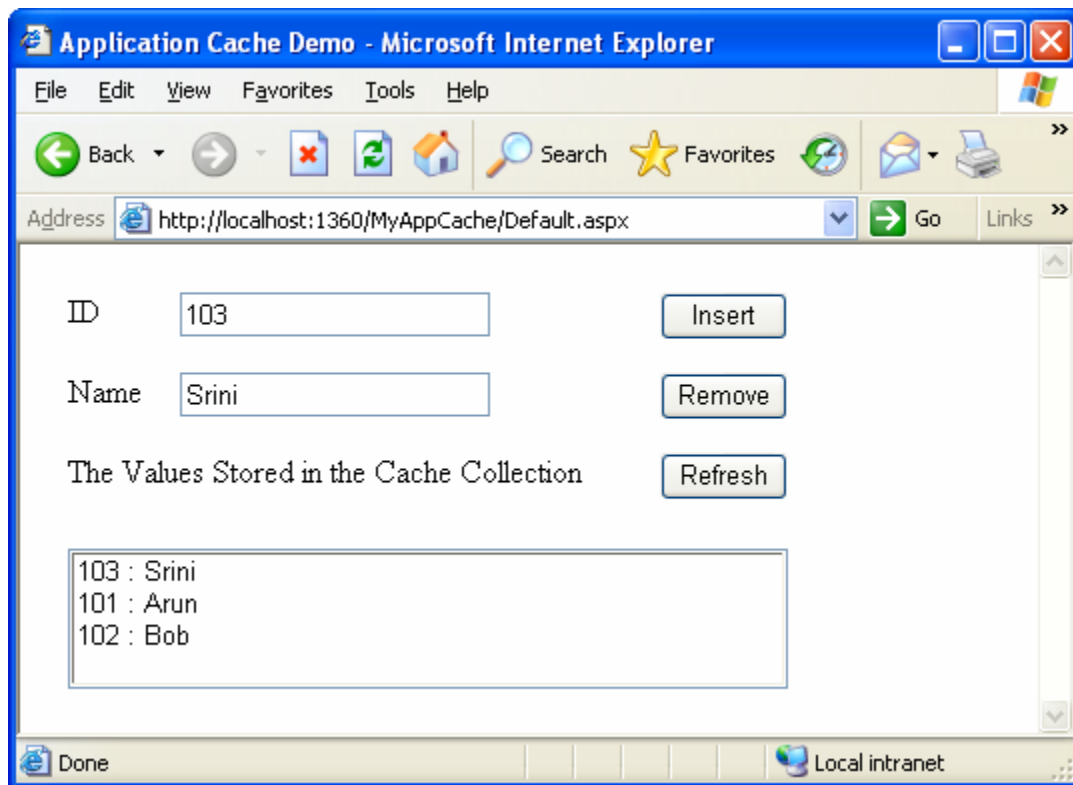
```
Cache.Remove("Value1", connectionString);
```

- *File and key dependencies* validates the cache item based on an external file. The Cache item is invalidated and eliminate from the cache, if the dependency changes.
- *Scavenging* means that the cache try to remove occasionally used items if memory is in short supply.
- By using *Expiration*, you can fix the lifetime of the item. E.g. we can terminate the item after 10 minutes of its last usage.

# Application Caching – Example

---

- A simple example to demonstrate the usage of *Cache.Insert()* and *Cache.Remove()* method.
- Source code for this example is present in *MyAppCache* directory.



- To remove an item, type it's ID in the textbox and click the Remove button.

# Application Caching – Example

## (Cont'd)

---

- In this example the data (ID, Name) entered is stored in the Cache using *Cache.Insert()* and *Cache.Remove()* methods.
- The values that are stored in the Cache are in the form of collections. So one can easily get the values stored in the collection as follows.

```
private void DisplayCache()
{
    string strCache = "";
    string strKey;

    listCache.Items.Clear();
    foreach(DictionaryEntry objItem in Cache)
    {
        strKey = objItem.Key.ToString();
        strCache = strKey + " : " +
            Cache[strKey].ToString();
        listCache.Items.Add(strCache);
    }
}
```

# Expiration

---

- **Absolute Cache Expiration:**

- The following code sets an absolute cache expiration time.

```
Cache.Insert("Value1", myInfo, null,  
    DateTime.Now.AddMinutes(2), TimeSpan.Zero);
```

- The parameter **DateTime.Now.AddMinutes(2)**, which indicates that the item expires 2 Minutes from the time it is inserted.
- The final argument, **TimeSpan.Zero** indicates that there is no relative expiration policy on this item.

- **Relative Cache Expiration:**

- The following code shows how to set a relative expiration policy.

```
Cache.Insert("Value1", myInfo,  
    DateTime.MaxValue, TimeSpan.FromMinutes(30));
```

- It inserts an item that expires 30 minutes after it is last accessed. Note the use of **DateTime.MaxValue**, which indicates that there is no absolute expiration policy on this item.
- You can also add an item to the cache with priority settings using the **priority** and **priorityDecay** parameters on the **Add()** or **Insert()** method.

# Problems in Caching

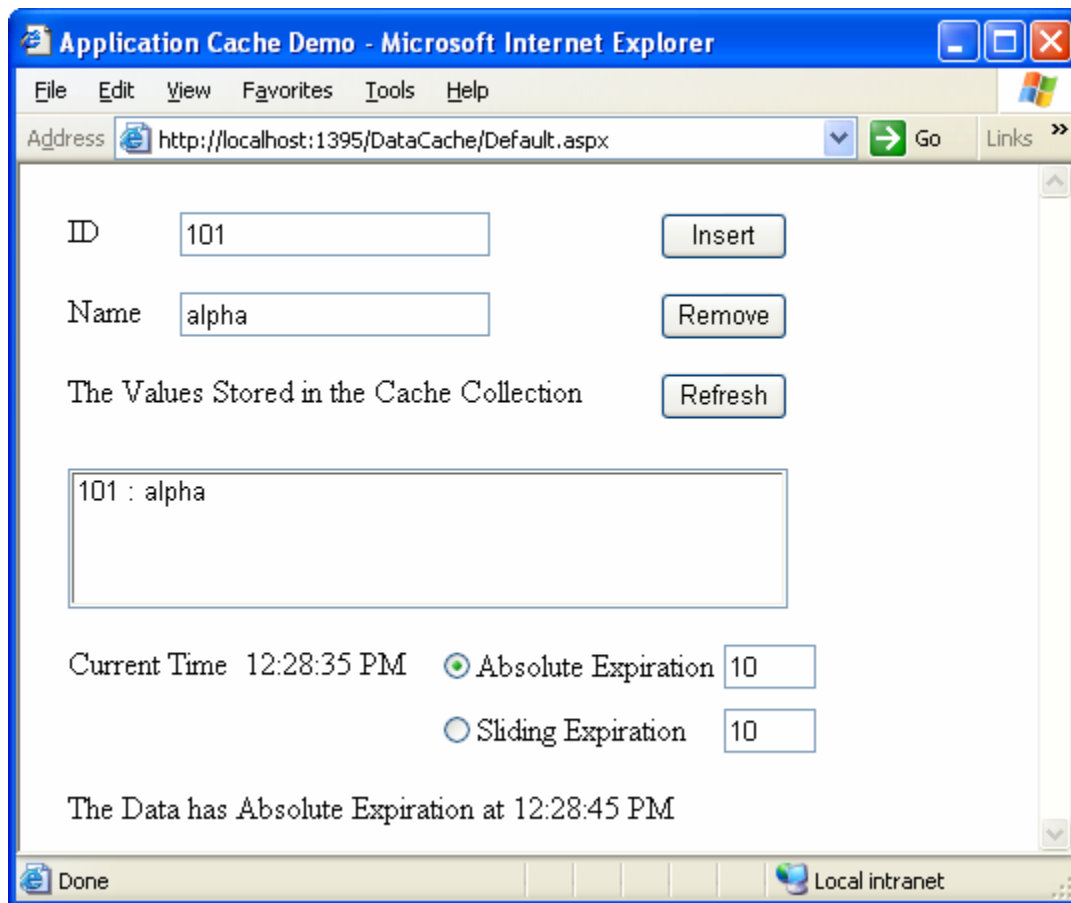
---

- **One of the key problems when designing caching strategies is discovering when the data in the cache should be removed from the cache.**
- **If the application updates the data at regular intervals, there is no problem. However, there may be other circumstances where the data is updated at relatively random intervals.**
- **In either case, the solution to the problem is to figure out the most favorable time interval for updating the cache, so that a balance can be achieved.**
- **You can include that information with your data, if you have figured out the optimal interval so that the caching systems can update their data appropriately.**

# Lab 7

## Data Caching (Absolute vs. Sliding Expiration)

In this lab you will enhance the MyAppCache program discussed previously in the chapter. The enhanced program adds the functionality of Expiration policies. You will gain practice working with both types of Expirations (Absolute & Sliding). You will also learn about the difference between these two types of Expiration policies.



Detailed instructions are contained in the Lab 7 write-up at the end of the chapter.

Suggested time: 30 minutes

## Summary

---

- **ASP.NET offers you a powerful, easy-to-use different caching mechanism that allows you to build Web applications with fast response and high throughput.**
- **In this age of business-to-business and business-to-consumer e-commerce, by means of ASP.NET caching, one can save the valuable processor time and resources.**
- **Output caching is a characteristic of ASP.NET that allows for the contents of an entire page to be stored in the Cache.**
- **Partial-Page Output Caching, or page fragment caching, allows particular portions of pages to be cached.**
- **ASP.NET provides you a solid, user-friendly caching mechanism that allows you to store expensive objects in memory across HTTP requests.**
- **Whether using Output Caching, Fragment Caching, or the Cache object directly, your Web applications will see spectacular enhancements in throughput and scalability through the technique of caching expensive dynamic data computations.**



## Lab 7

### Data Caching (Absolute vs. Sliding Expiration)

#### Introduction

In this lab you will enhance the **MyAppCache** program discussed previously in the chapter. The enhanced program adds the functionality of Expiration policies. You will gain practice working with both types of Expirations (Absolute & Sliding). You will also learn about the difference between these two types of Expiration policies.

**Suggested Time:** 30 minutes

**Root Directory:** OIC\AspCs

**Directories:** Labs\Lab7\DataCache (do your work here)  
 Chap07\MyAppCache (backup of starter code)  
 Chap07\DataCache (answer)

#### Instructions:

1. Add new controls to the main form as shown.

The screenshot shows a web form with the following controls:

- An input field labeled **ID** with a small green icon to its left.
- An input field labeled **Name** with a small green icon to its left.
- A button labeled **Insert** with a small green icon to its left.
- A button labeled **Remove** with a small green icon to its left.
- A text box containing the text "The Values Stored in the Cache Collection" with a small green icon to its left.
- A button labeled **Refresh** with a small green icon to its left.
- A large text area labeled **Unbound** with a small green icon to its left.
- A label **Current Time** with a small green icon to its left, followed by an input field labeled **[blTime]** with a small green icon to its left.
- A radio button labeled **Absolute Expiration** with a small green icon to its left, followed by an input field with a small green icon to its left.
- A radio button labeled **Sliding Expiration** with a small green icon to its left, followed by an input field with a small green icon to its left.
- An input field labeled **[blExpires]** with a small green icon to its left.

2. Set the Checked property of the Absolute Expiration radio button to **True**. Set the **GroupName** property of both radio buttons to **Expiration**. Assigning these radio buttons the same group name will ensure that when one is checked at runtime, the other will be automatically unchecked.

3. Provide **Text** values of 10 for the two expiration textboxes.
4. Add code to display the time, both on page load and refresh.
5. Check which Radio button is checked and based on that write the code separately for Absolute Expiration and Sliding Expiration such as:

#### Absolute Expiration:

```
Cache.Insert(txtID.Text, txtName.Text, null,
    DateTime.Now.AddSeconds(Convert.ToDouble(txtAbsolute.Text)),
    TimeSpan.Zero);
DisplayCache();
lblExpires.Text = "The Data has Absolute Expiration at " +
    DateTime.Now.AddSeconds(Convert.ToDouble(
    txtAbsolute.Text)).ToLongTimeString();
```

#### Sliding Expiration:

```
Cache.Insert(txtID.Text, txtName.Text, null, DateTime.MaxValue,
    TimeSpan.FromSeconds(Convert.ToDouble(txtSlide.Text)));
DisplayCache();
lblExpires.Text = "The Data has Sliding Expiration at " +
    DateTime.Now.AddSeconds(Convert.ToDouble(
    txtSlide.Text)).ToLongTimeString();
```

#### Remarks:

- The **Convert.ToDouble** method is used to convert any valid numeric or string expression to **double** data type.
- After setting either Absolute Expiration or Sliding Expiration, **lblExpire** shows the time at which expiration occurs.
- You can check the Content of Cache Object using the **DisplayCache ()** function.
- In the output you can verify that Absolute Expiration is expired after a defined absolute time.
- The expiration time of Sliding Expiration extends or slides when the page is again refreshed within that defined time, and expire time is again reset on a sliding basis.
- It is easy to observe cache expiration by setting a brief expiration time and clicking the Refresh button.

#### Checking for Null Items in Cache:

6. When an item has expired from the cache, you may hit an exception in the **DisplayCache** method. The expression **Cache[key]** returns **null** if there is no item in

the cache corresponding to this key. So you should not use **Cache[key]** until you check if it is not null. Thus the **DispalyCache** method should be rewritten to make such a test.

```
strCache = strKey + " : ";  
if (Cache[strKey] != null)  
    strCache += Cache[strKey].ToString();  
listCache.Items.Add(strCache);
```



## **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.**
- **Attach the debugger to a running ASP.NET application that was not started 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 created using Visual Studio is especially easy, but the debugger can also be attached to ASP.NET applications created without 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

---

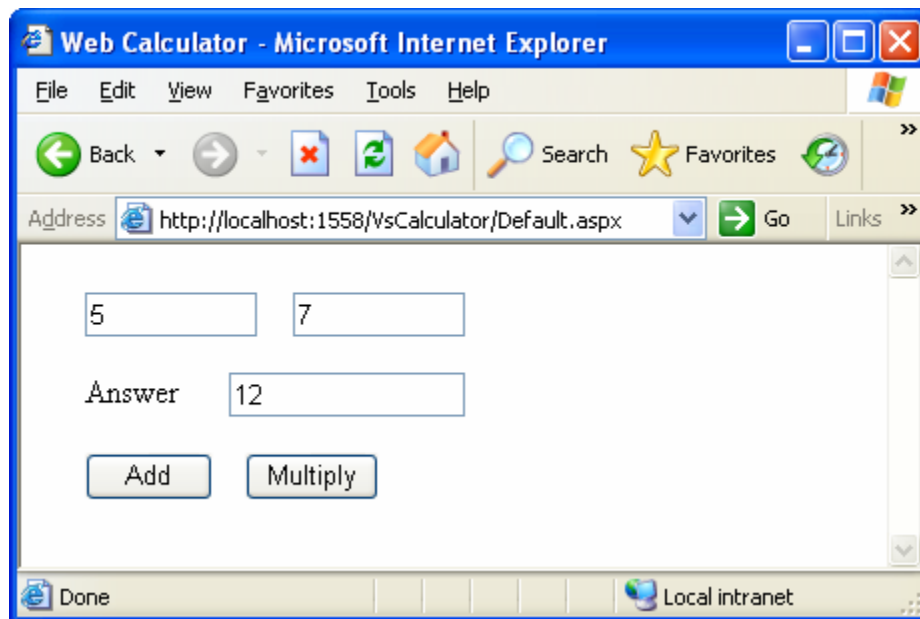
- **Applications created using Visual Studio are very easy to debug.**
  - Build in Debug mode (the default).
  - You can then set breakpoints, single-step, use watch windows, and all the other features of the Visual Studio debugger.



# Calculator Example

---

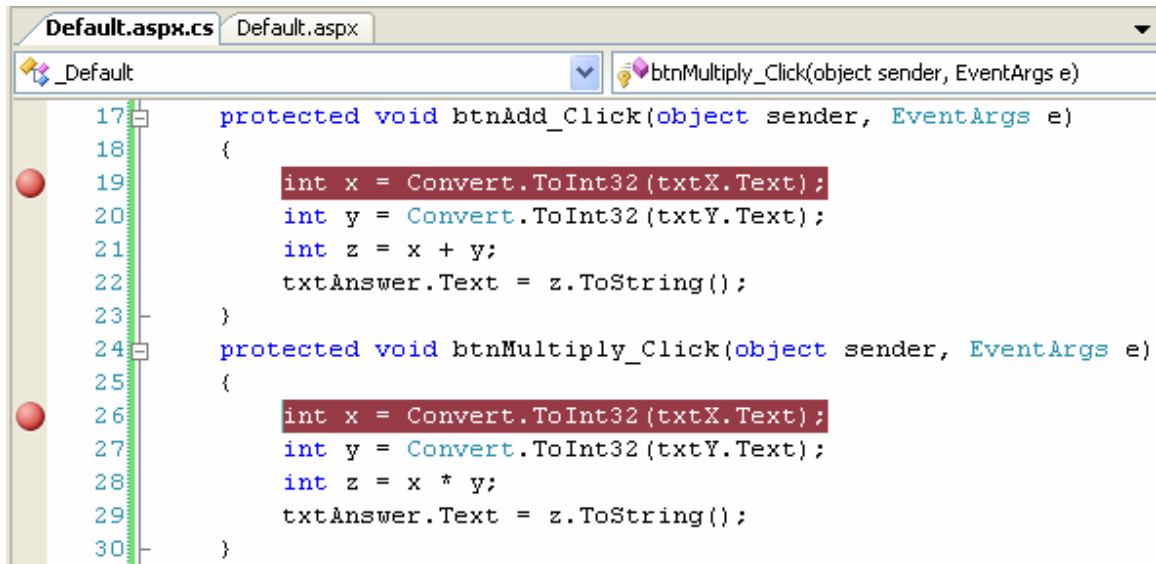
- We will illustrate diagnostics in this chapter with a simple Web calculator program *VsCalculator*.




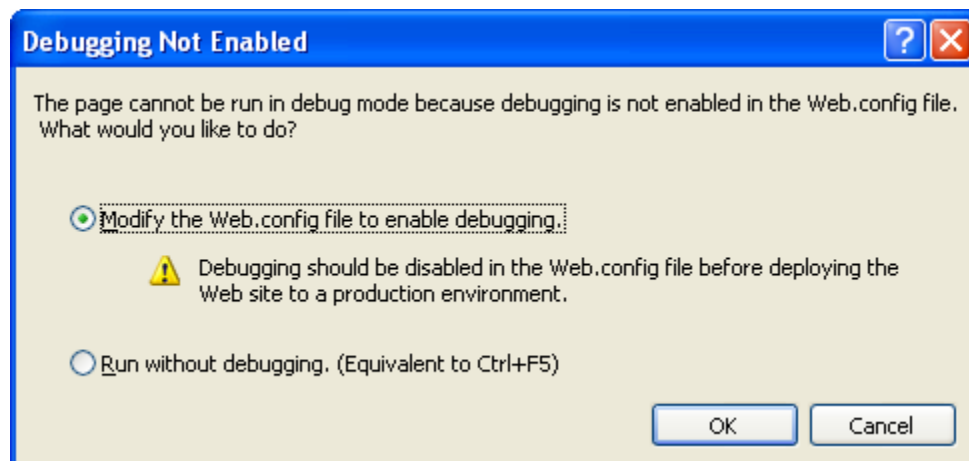
- An initial Add works fine, but if you try Multiply or adding different numbers, the application behaves strangely.
- In the demo on the following page, your results may vary depending on how long it takes you to perform the various steps.

# Debugging Calculator

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



2. Run under the debugger (Debug | Start or F5 or the toolbar button ). You will see a dialog box asking if you want to enable debugging.




## Debugging Calculator (Cont'd)

---

3. Examine the **Web.Config** file that is created for you.

```
<configuration>
  <appSettings/>
  <connectionStrings/>
  <system.web>
    <!--
    Set compilation debug="true" to insert debugging
    symbols into the compiled page. Because this
    affects performance, set this value to true only
    during development.
    -->
    <compilation debug="true"/>
    ...
```

4. Enter some numbers to add, and click the Add button. You should hit a breakpoint. Continue (F5 or toolbar button .
5. Click the Multiply button. You don't hit the breakpoint!
6. 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.**

```
<!-- APPLICATION-LEVEL TRACE LOGGING
Application-level tracing enables trace log
output for every page within an application.
Set trace enabled="true" to enable application
trace logging. If pageOutput="true", the
trace information will be displayed at the
bottom of each page. Otherwise, you can view
the application trace log by browsing the
"trace.axd" page from your web application
root.
-->
<trace enabled="true"
    requestLimit="10"
    pageOutput="false"
    traceMode="SortByTime"
    localOnly="true"
/>
```

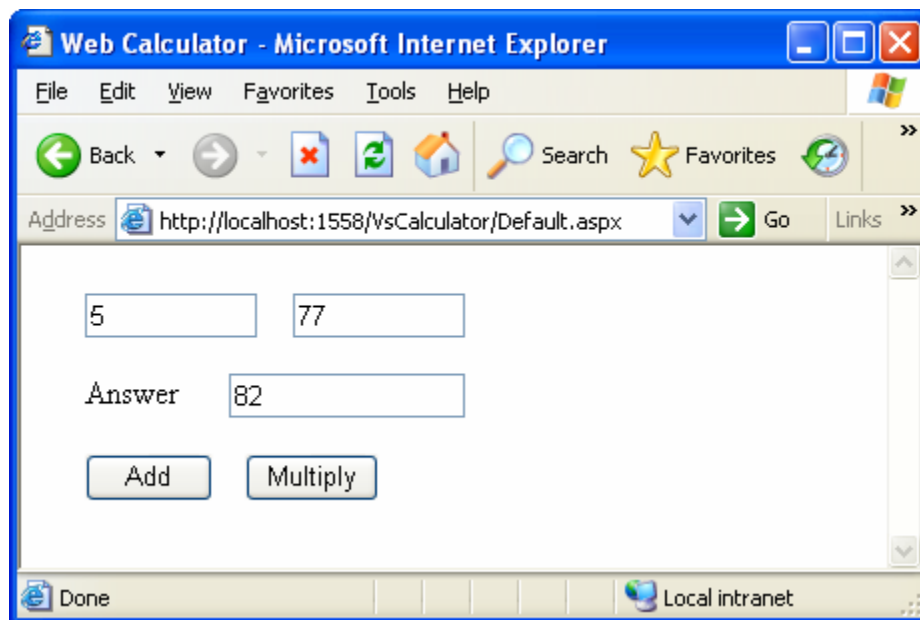
- As the comment says, you can either have the output shown directly on the page, or else sent to a trace log that can be viewed by browsing the **trace.axd** page.

# Tracing Calculator

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

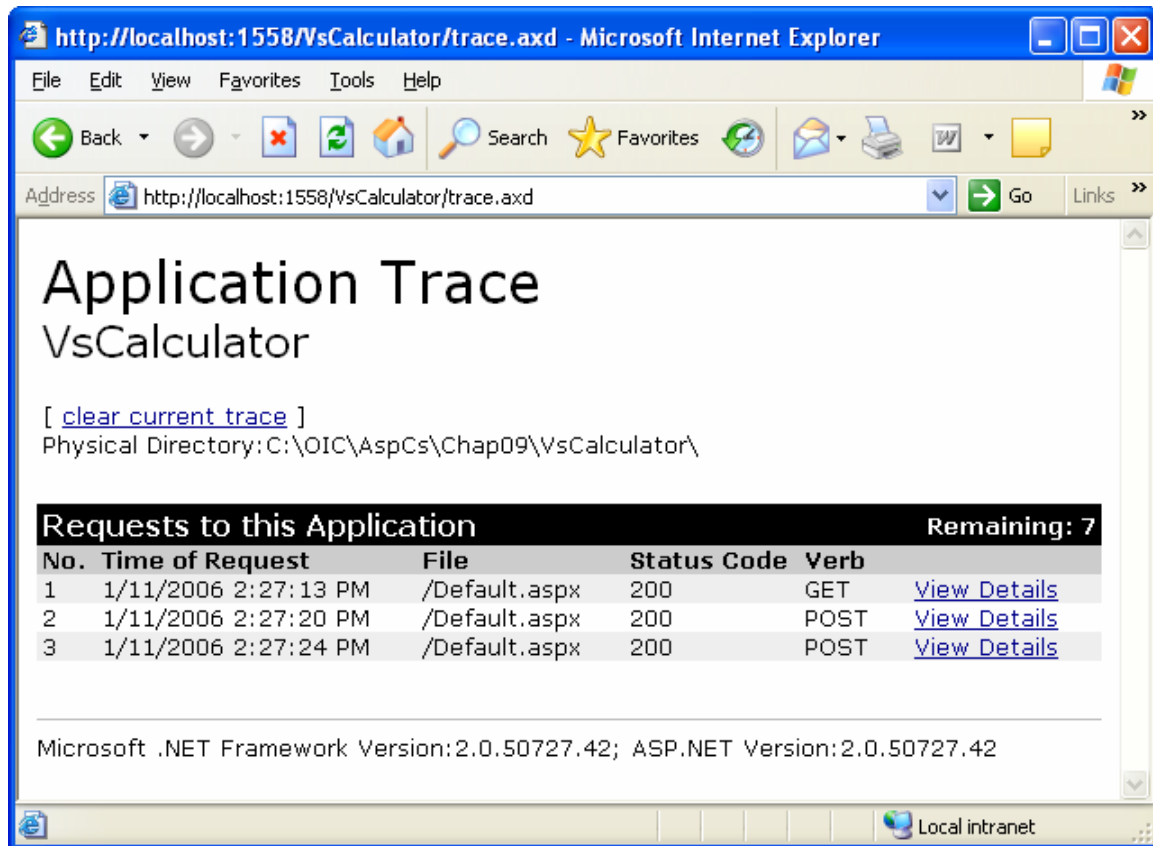
```
<compilation debug="true"/>  
<trace enabled="true"/>
```

2. Run the application, not in the debugger.
3. Enter the numbers 5 and 77 and click the Add button.
4. 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)

5. To see the trace, navigate in the browser to the **trace.axd** URL at the root of the application directory.



6. 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)

---

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

Form Collection	
Name	Value
__VIEWSTATE	/wEPDwULLTExMTI5OTk1ODVkZFX96mWoiBbySoPXmrUpAnAaAj/k
txtX	5
txtY	77
txtAnswer	
btnAdd	Add
__EVENTVALIDATION	/wEWBgLzljQUAQKShpCYCAKShvy8DwLdsKuIBALsmoXRCQKMk4HYI

8. View the details of the second POST.

Form Collection	
Name	Value
__VIEWSTATE	/wEPDwULLTExMTI5OTk1ODVkZFX96mWoiBbySoPXmrUpAnAaAj/k
txtX	5
txtY	777
txtAnswer	82
btnAdd	Add
__EVENTVALIDATION	/wEWBgLzljQUAQKShpCYCAKShvy8DwLdsKuIBALsmoXRCQKMk4HYI

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

# Using the Page Cache

---

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

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

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

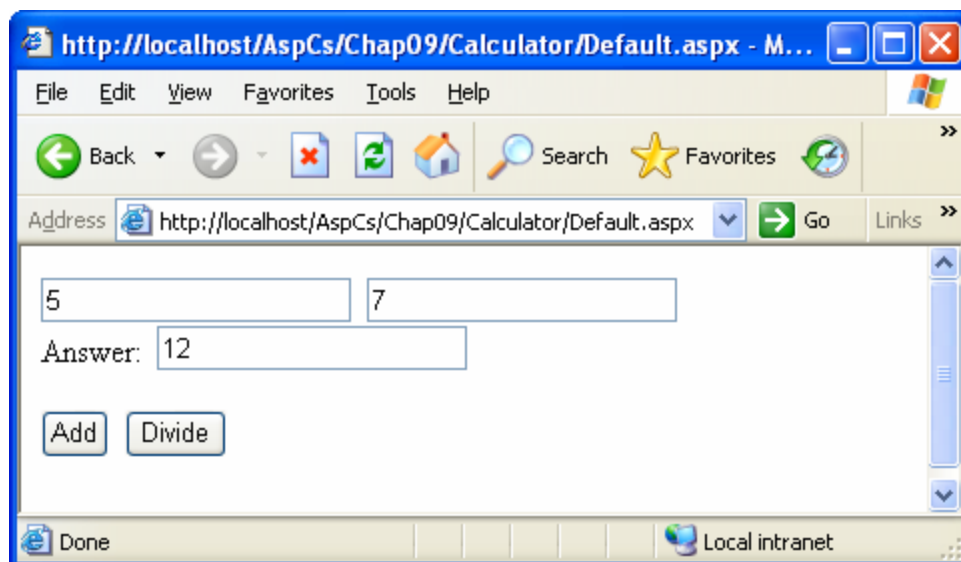
```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Web Calculator</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:TextBox ID="txtX" runat="server"
Style="z-index: 100; left: 32px; position:
absolute; top: 24px" Width="80px">
</asp:TextBox>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

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



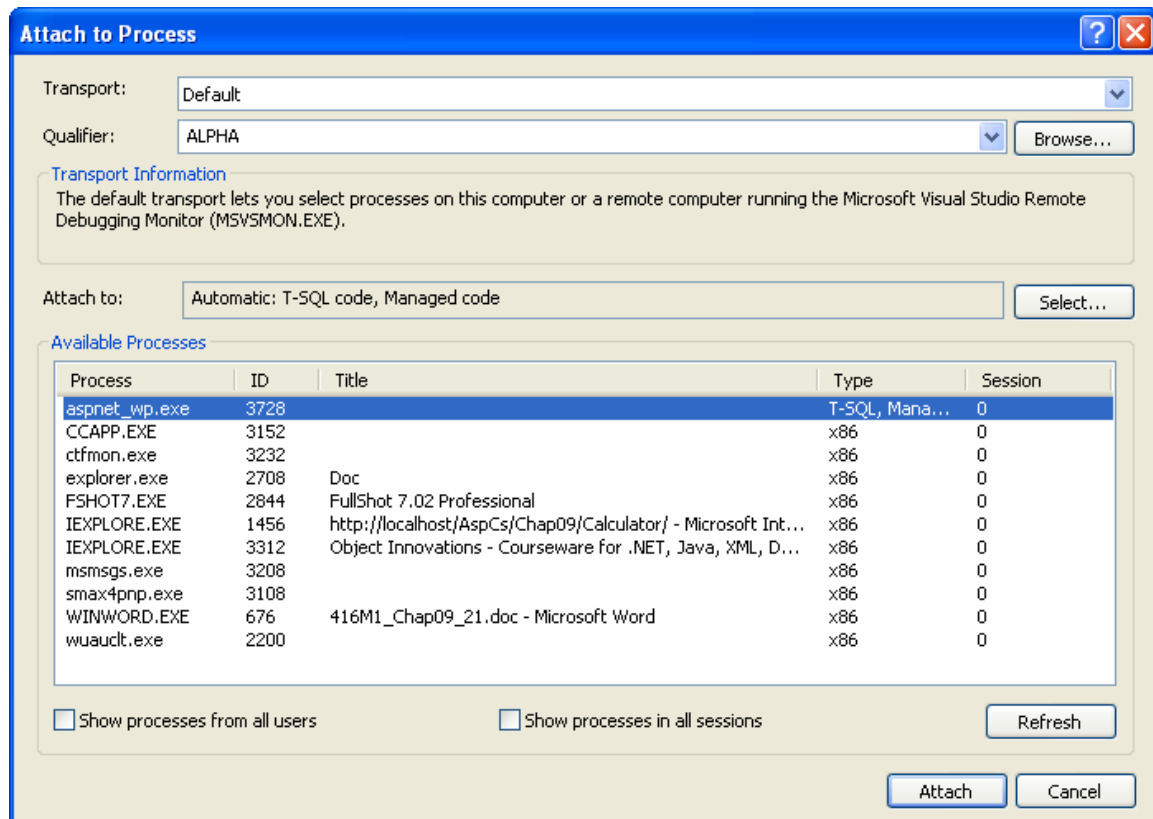
# An ASP.NET Page Without Visual Studio

- While it is seamless to build and debug from within Visual Studio, you can also attach the Visual Studio debugger to an ASP.NET application not created using Visual Studio.
  - We need full-blown Visual Studio for this, not an Express Edition.
- 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.



# Attaching to VS Debugger

- **Let's demonstrate attaching the Visual Studio debugger to a running process.**
1. View the **Calculator/Default.aspx** page in the browser and leave the window open.
  2. In Visual Studio select the Tools | Attach to Process menu. In the Processes dialog, select the aspnet\_wp.exe process.

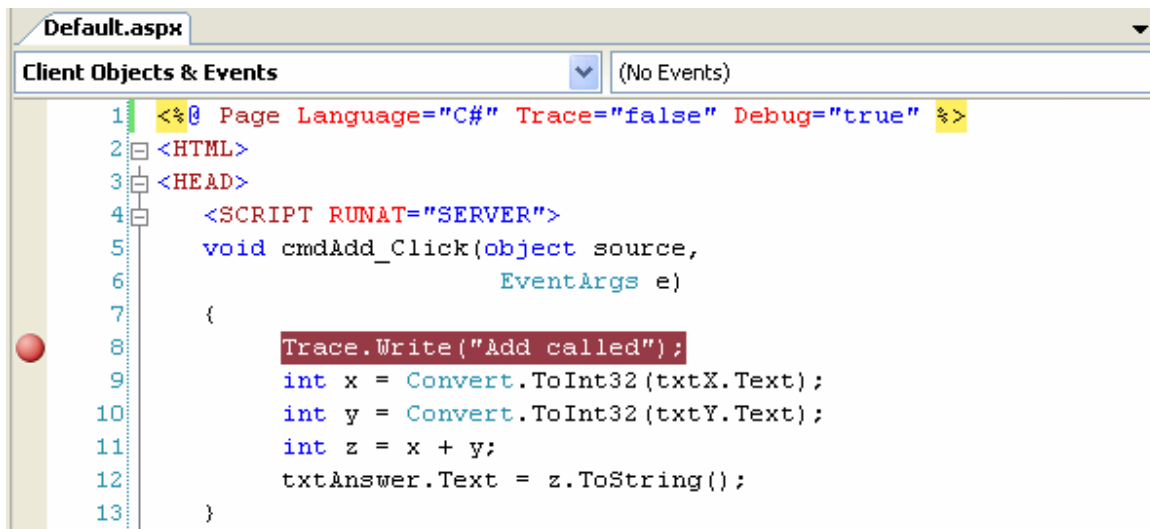


3. Click Attach ...

## Attaching to VS Debugger (Cont'd)

---

4. In Visual Studio open up the file **Calculator/Default.aspx**. Set a breakpoint at the first instruction of the handler of the Add button.



The screenshot shows the Visual Studio IDE with the file **Default.aspx** open. The **Client Objects & Events** window is visible, showing **(No Events)**. The code in the editor is as follows:

```
1 <%@ Page Language="C#" Trace="false" Debug="true" %>
2 <HTML>
3 <HEAD>
4 <SCRIPT RUNAT="SERVER">
5 void cmdAdd_Click(object source,
6     EventArgs e)
7 {
8     Trace.Write("Add called");
9     int x = Convert.ToInt32(txtX.Text);
10    int y = Convert.ToInt32(txtY.Text);
11    int z = x + y;
12    txtAnswer.Text = z.ToString();
13 }
```

A red circular breakpoint is set on line 8, at the start of the `Trace.Write("Add called");` statement.

5. Back in the browser window, enter two numbers and click Add. You should now hit your breakpoint. You should be able to examine the values of variables, single-step, and so on. You are debugging!

# 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"
    ErrorPage="PageError.html" %>
<HTML>
<HEAD>
    <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" />
        ...
    </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.

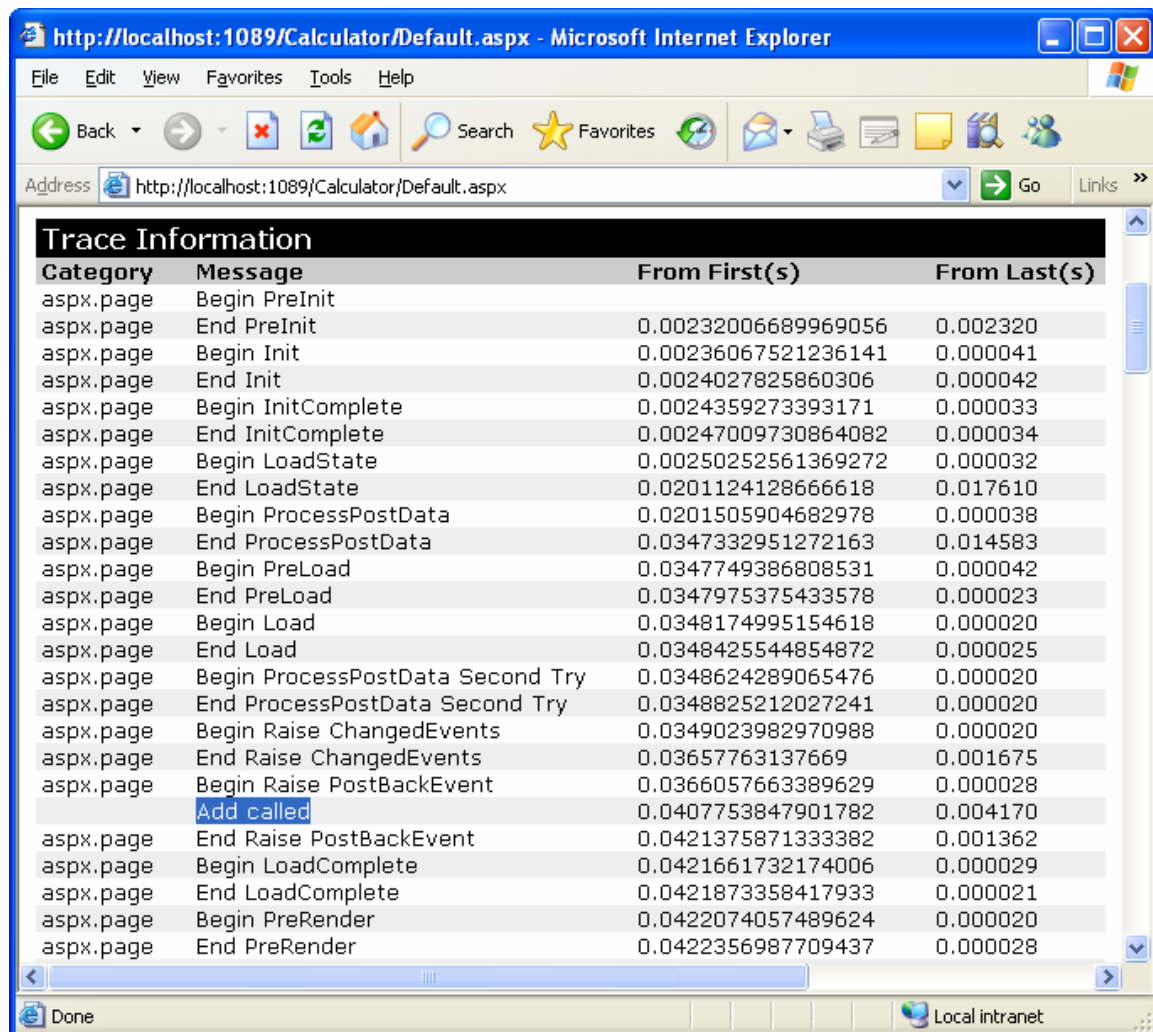
```
...  
<SCRIPT RUNAT="SERVER">  
void cmdAdd_Click(object source,  
                    EventArgs e)  
{  
    Trace.Write("Add called");  
    int x = Convert.ToInt32(txtX.Text);  
    int y = Convert.ToInt32(txtY.Text);  
    int z = x + y;  
    txtAnswer.Text = z.ToString();  
}  
...
```

- 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.



Trace Information

Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	0.00232006689969056	0.002320
aspx.page	Begin Init	0.00236067521236141	0.000041
aspx.page	End Init	0.0024027825860306	0.000042
aspx.page	Begin InitComplete	0.0024359273393171	0.000033
aspx.page	End InitComplete	0.00247009730864082	0.000034
aspx.page	Begin LoadState	0.00250252561369272	0.000032
aspx.page	End LoadState	0.0201124128666618	0.017610
aspx.page	Begin ProcessPostData	0.0201505904682978	0.000038
aspx.page	End ProcessPostData	0.0347332951272163	0.014583
aspx.page	Begin PreLoad	0.0347749386808531	0.000042
aspx.page	End PreLoad	0.0347975375433578	0.000023
aspx.page	Begin Load	0.0348174995154618	0.000020
aspx.page	End Load	0.0348425544854872	0.000025
aspx.page	Begin ProcessPostData Second Try	0.0348624289065476	0.000020
aspx.page	End ProcessPostData Second Try	0.0348825212027241	0.000020
aspx.page	Begin Raise ChangedEvents	0.0349023982970988	0.000020
aspx.page	End Raise ChangedEvents	0.03657763137669	0.001675
aspx.page	Begin RaisePostBackEvent	0.0366057663389629	0.000028
aspx.page	Add called	0.0407753847901782	0.004170
aspx.page	End RaisePostBackEvent	0.0421375871333382	0.001362
aspx.page	Begin LoadComplete	0.0421661732174006	0.000029
aspx.page	End LoadComplete	0.0421873358417933	0.000021
aspx.page	Begin PreRender	0.0422074057489624	0.000020
aspx.page	End PreRender	0.0422356987709437	0.000028

## 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");
    ...
}
```

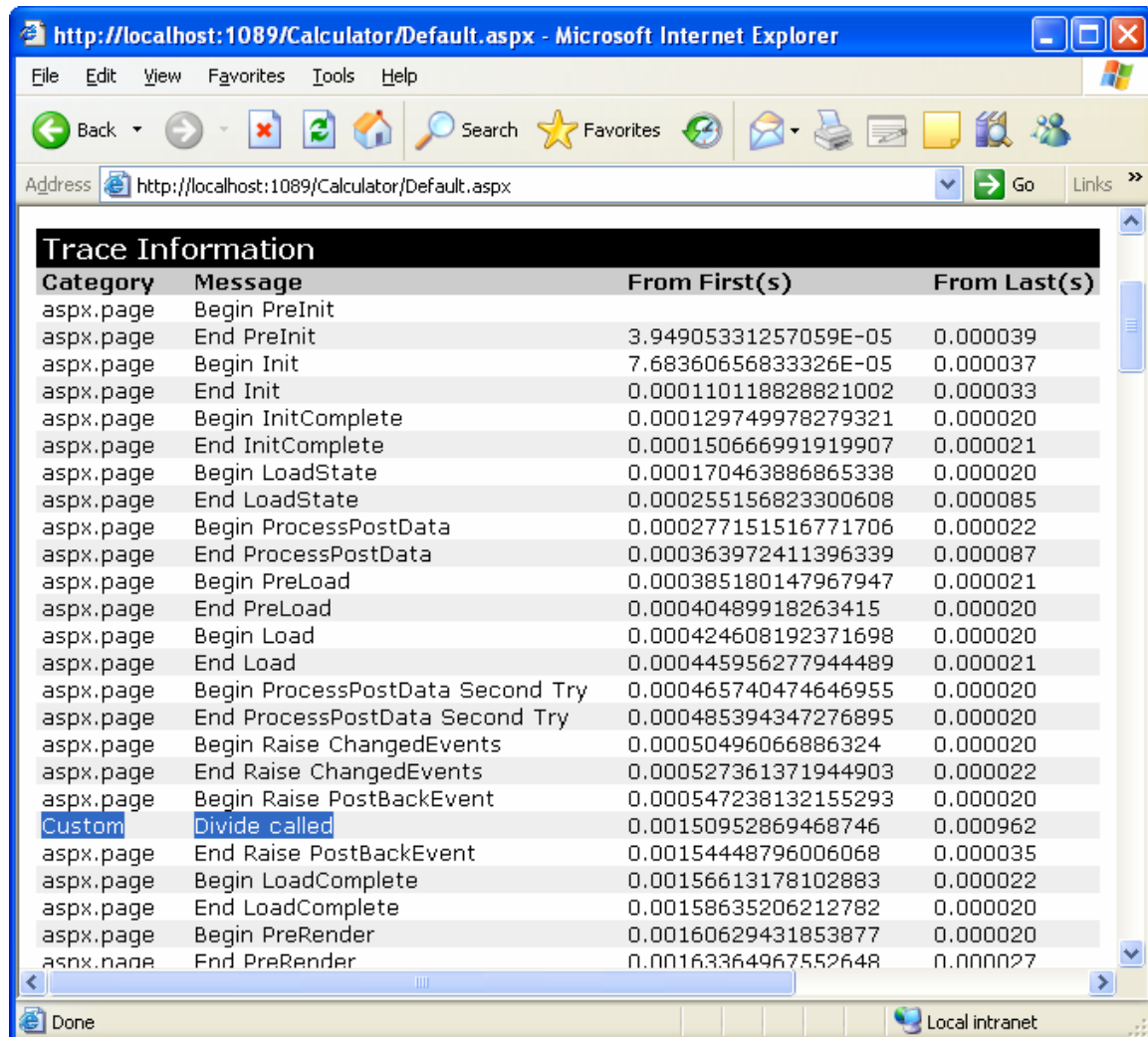
- You can verify by placing a breakpoint on the **Trace.Write()** statement, with debugging enabled and tracing disabled.
- **What this means is that you can embed useful trace statements in a production application without concern about output or computation time in the normal case when tracing is disabled.**
  - 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.

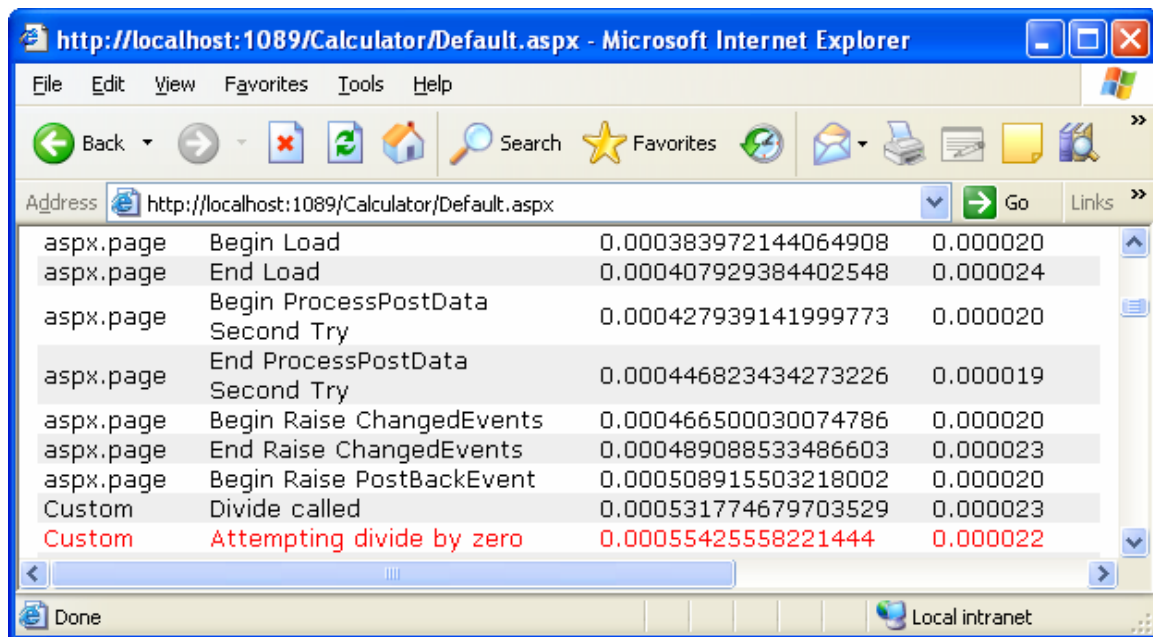




# Trace Warning

- 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.

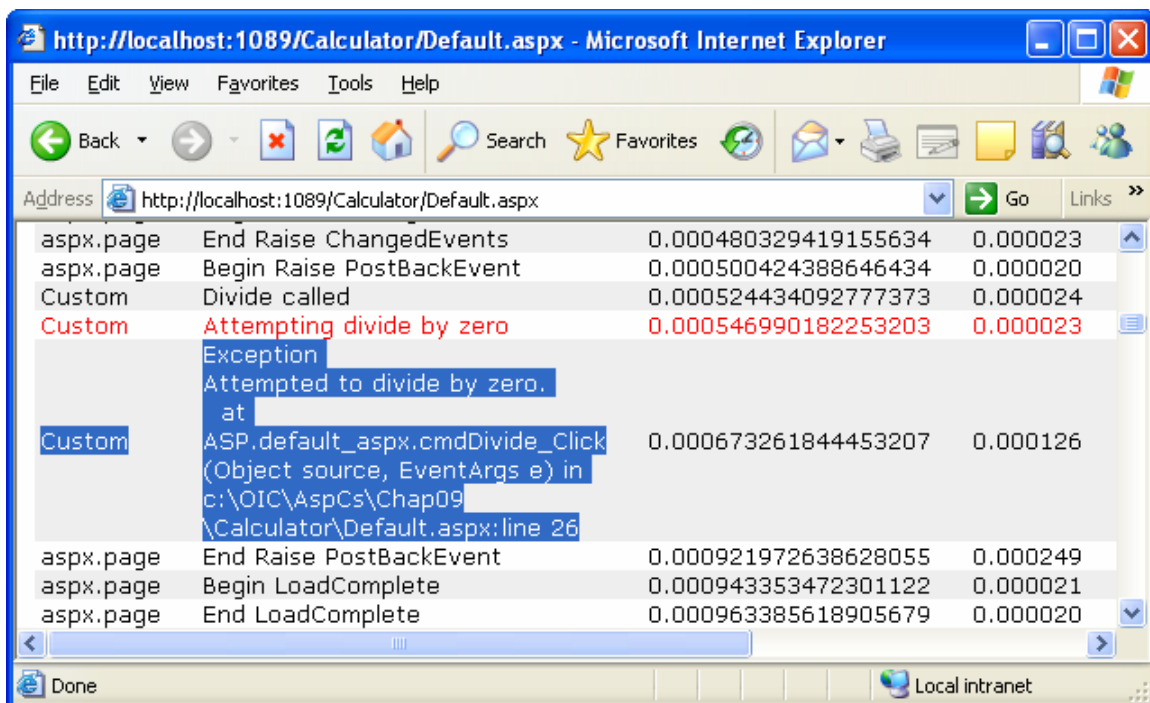
```
try
{
    int x = Convert.ToInt32(txtX.Text);
    int y = Convert.ToInt32(txtY.Text);
    if (y == 0)
        Trace.Warn("Custom",
                    "Attempting divide by zero");
    ...
}
```



# 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);
}
```



## Errors in ASP.NET

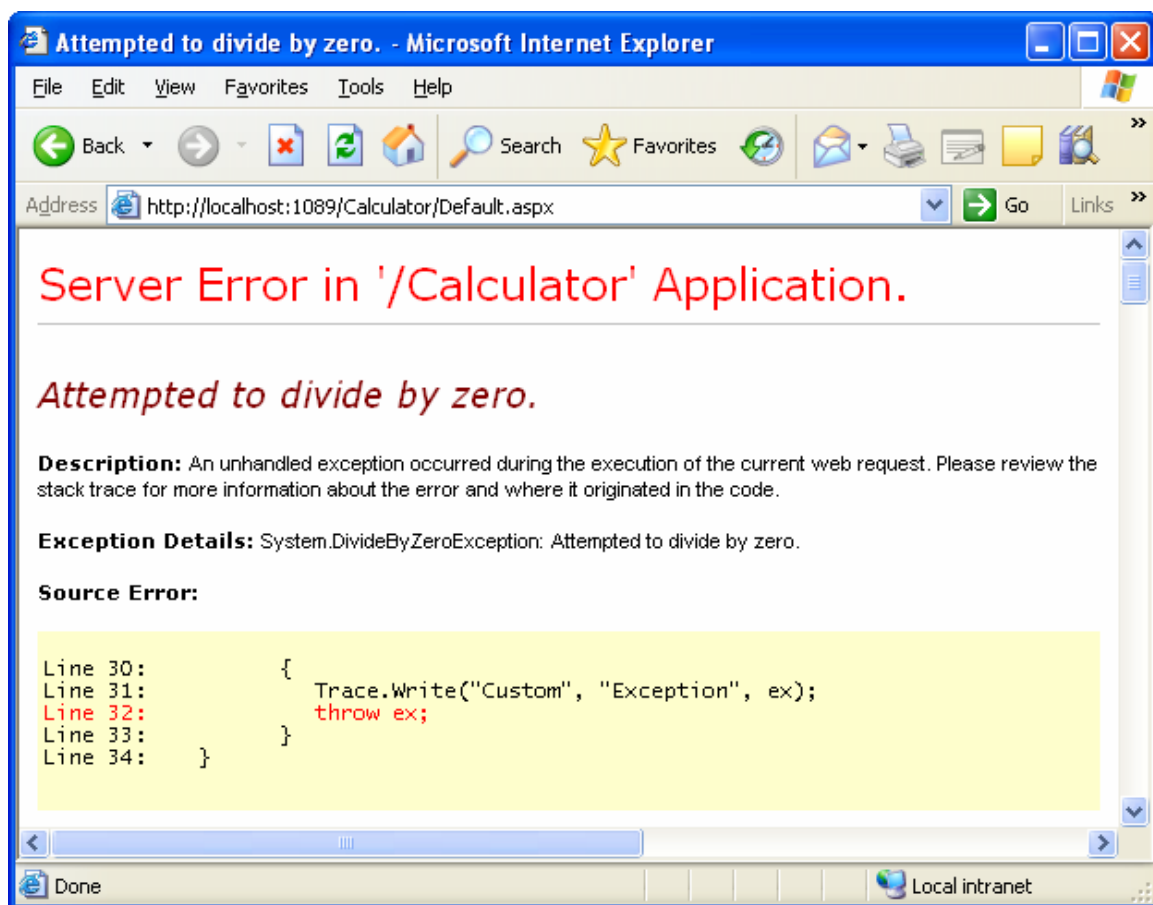
---

- **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.**
- **You can attach the debugger to running ASP.NET applications that were not started using Visual Studio.**
  - You need the full-blown version of Visual Studio.
- **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

<b>Directories:</b>	<b>Labs\Lab9\Error</b>	(do your work here)
	<b>Chap09\Calculator</b>	(contains backup of starter page)
	<b>Chap09\Error</b>	(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. Create a **Web.config** file with 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



# **Chapter 14**

## **Introduction to ASP.NET AJAX**

# Introduction to ASP.NET AJAX

## Objectives

---

*After completing this unit you will be able to:*

- **Position rich Internet applications in the spectrum of desktop and Web applications.**
- **Describe the use of JavaScript to implement rich client applications.**
- **Describe AJAX, or Asynchronous JavaScript and XML.**
- **Describe the use of partial page rendering in AJAX applications to enhance the user's experience.**
- **Use the ScriptManager and UpdatePanel controls to implement AJAX functionality such as partial page rendering.**
- **Use the AJAX Client Library to add client-side AJAX functionality to your Web applications.**
- **Describe the AJAX Control Toolkit and use it to add rich client functionality to your Web applications.**

# Desktop Applications

---

- **There are two kinds of desktop applications:**
  - Standalone personal productivity applications running on a PC.
  - Networked applications in which the desktop PC is a client for code running on a server, possibly through multiple tiers.
- **Advantages:**
  - A rich user interface experience leveraging the advanced pointing device and graphics capabilities of the PC operating system.
  - Ability to run disconnected from a network and to make use of local computer resources, such as hard disk storage.
- **Disadvantages:**
  - The application is tied to a particular PC operating system and will not run in a different environment.
  - There is significant administrative overhead in installing desktop applications and in making sure they are kept up-to-date.
- **Also, there are configuration issues that the individual user has to cope with.**
  - Have you had a Windows application start acting strangely, perhaps because some Registry or other configuration information has become corrupted?

# Web Applications

---

- **Web applications are accessible through virtually any browser running on any PC operating system.**
- **A Web application is trivial to deploy, as it just needs to be posted on a website, and any user can run it merely by pointing the browser to a certain URL.**
- **But traditionally Web applications have a much more primitive user interface, with much less interactivity with the user.**
- **Typical user interactions are quite simple:**
  - A page may simply display some information, with links to go to other pages.
  - A page may have a simple form which the user can fill out and submit.
  - The response is another page.

# Plug-Ins

---

- **The user interface provided by standard browsers is fairly limited, and early on browsers began implementing various plug-in technologies.**
  - A plug-in is a piece of code that downloads from the server and augments the functionality of the browser.
- **Classic examples of plug-ins:**
  - Netscape Navigator plug-ins
  - Microsoft ActiveX controls
  - Java applets
- **Modern examples of the plug-in approach:**
  - Flash
  - Microsoft Silverlight (formerly known as WPF/E)
  - JavaFX from Sun

# Client-Side Scripting

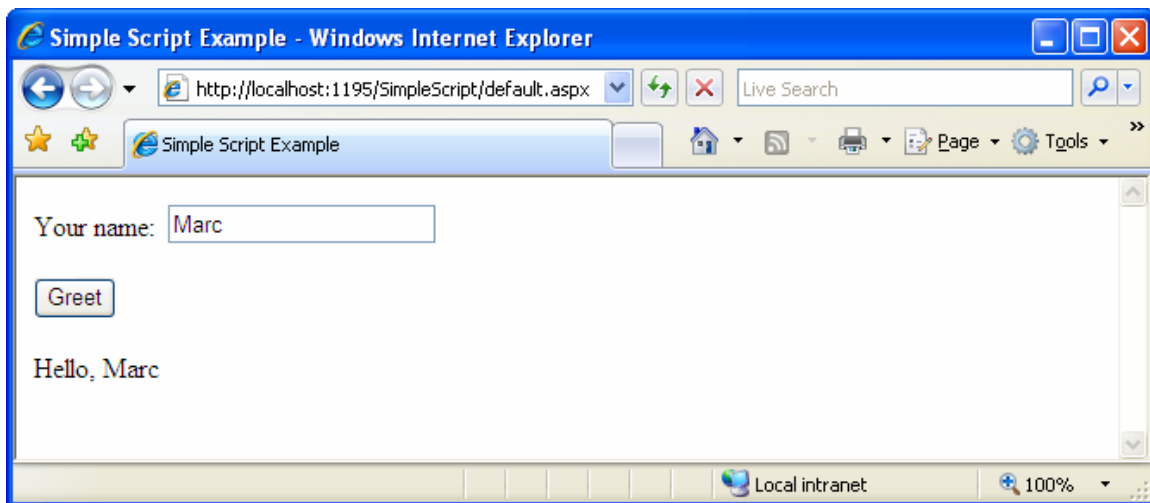
---

- **Another approach to enhancing Web applications is client-side scripting.**
- **HTML provides a `<script>` tag, which can be used to specify code that will execute in response to events in the browser.**
- **There are various scripting languages that can be used, but by far the most popular is *JavaScript*.**
  - JavaScript was originally introduced by Netscape.
  - Despite its name and syntax, it has nothing to do with Java!
  - Microsoft's version of JavaScript is called JScript.
- **JavaScript has been standardized by ECMA; the standard version is known as *ECMAScript*.**
  - The stable version is ECMAScript Edition 3, which was finalized in 1999.
  - After a long hiatus, there is active work currently going on for ECMAScript Edition 4. See:  
<http://www.ecmascript-lang.org/>
- **A scripting language for Internet Explorer is *VBScript*.**
  - But VBScript is only available on Microsoft platforms and so does not have the reach of JavaScript.

# JavaScript Example

---

- We will use JavaScript in this chapter, so let's look at a simple example right away.
  - In Visual Studio 2008 open the website **SimpleScript** in **c:\OIC\AspCs\Chap14**.
  - Run the program from within Visual Studio.



- Now try leaving the “Your name” field blank. You will see an error message displayed in a message box.



- Note that this validation occurs *completely on the client*, with no postback to the server involved.

# Script Code

---

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>Simple Script Example</title>
<script type="text/javascript">

function btnGreet_onclick()
{
    var name =
        document.getElementById("txtName").value;
    if (name.length == 0)
    {
        alert("Please enter your name");
        return false;
    }
    else
        return true;
}

</script>
</head>
<body>
    <form id="form1" runat="server"
        onsubmit="return btnGreet_onclick();">
    ...

```

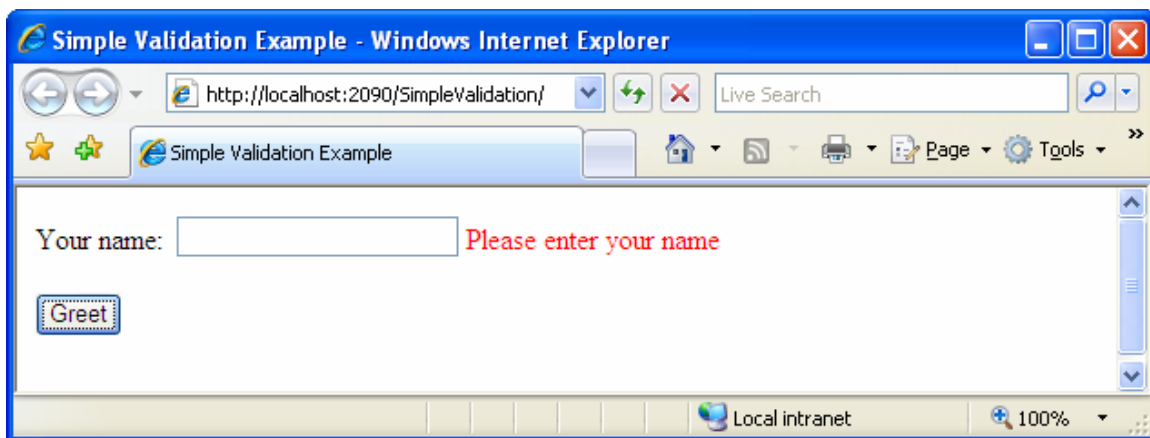
- Since the **btnGreet\_onclick()** function returns **false** when the name has length zero, the form will **not** be submitted to the server.



# JavaScript in ASP.NET

---

- **ASP.NET uses JavaScript under the hood a great deal in order to achieve many of its effects.**
- **As an example, consider the same program using the *RequiredFieldValidator* server control**
  - See **SimpleValidation** in the **Chap14** directory.



- You may examine the generated HTML/JavaScript using Page | View Source in Internet Explorer 7 or View | Source in IE 6.

```
...  
function ValidatorOnSubmit() {  
    if (Page_ValidationActive) {  
        return ValidatorCommonOnSubmit();  
    }  
    else {  
        return true;  
    }  
}
```

# Dynamic Pages

---

- **The original Web primarily served static and non-modifiable HTML pages.**
  - These pages were primarily text with a little graphics, but as bandwidth increased Web pages became more complex, with far more graphics, streaming audio, and video.
- **Beyond serving static pages, Web applications were developed that allowed the server to create pages on the fly.**
  - Popular server-side technologies included CGI (common gateway interface), ASP (Active Server Pages), JSP (JavaServer Pages) and PHP and so on.
- **In the late 1990s Dynamic HTML (DHTML) was introduced by the browser vendors.**
  - The World Wide Web Consortium (W3C) standardized a Document Object Model (DOM) for page content.
  - This means that browsers can expose the whole content of a page through a read/write object model.
- **Hence, it is now feasible to dynamically alter the appearance of a Web page through client-side code.**
- **Another modern aspect of Web page development is the use of XHTML, which has stricter syntax than traditional HTML.**
  - Visual Studio will generate XHTML compliant pages for us.

## Efficient Page Redraws

---

- **The normal way a page is updated is by bringing a whole new page down from the server.**
  - This works fine for simple text and a little graphics.
  - But what about complex pages with significant graphics, elaborate data in tables, and so on?
- **A more efficient approach is to bring down only the data that changes, and let the client dynamically update the page through DHTML/DOM.**
- **Technology to do this, called *Remote Scripting* (RS) was introduced by Microsoft in 1997.**
  - RS originally used a Java applet and later an ActiveX object `XMLHttpRequest`.
- **Modern browsers implement the *XMLHttpRequest* object directly, providing a uniform approach to obtaining selected data from the server to facilitate partial page redraws.**
  - Mozilla has always used `XMLHttpRequest`.
  - Internet Explorer before 7.0 used an ActiveX object, but from 7.0 on implements `XMLHttpRequest` directly.

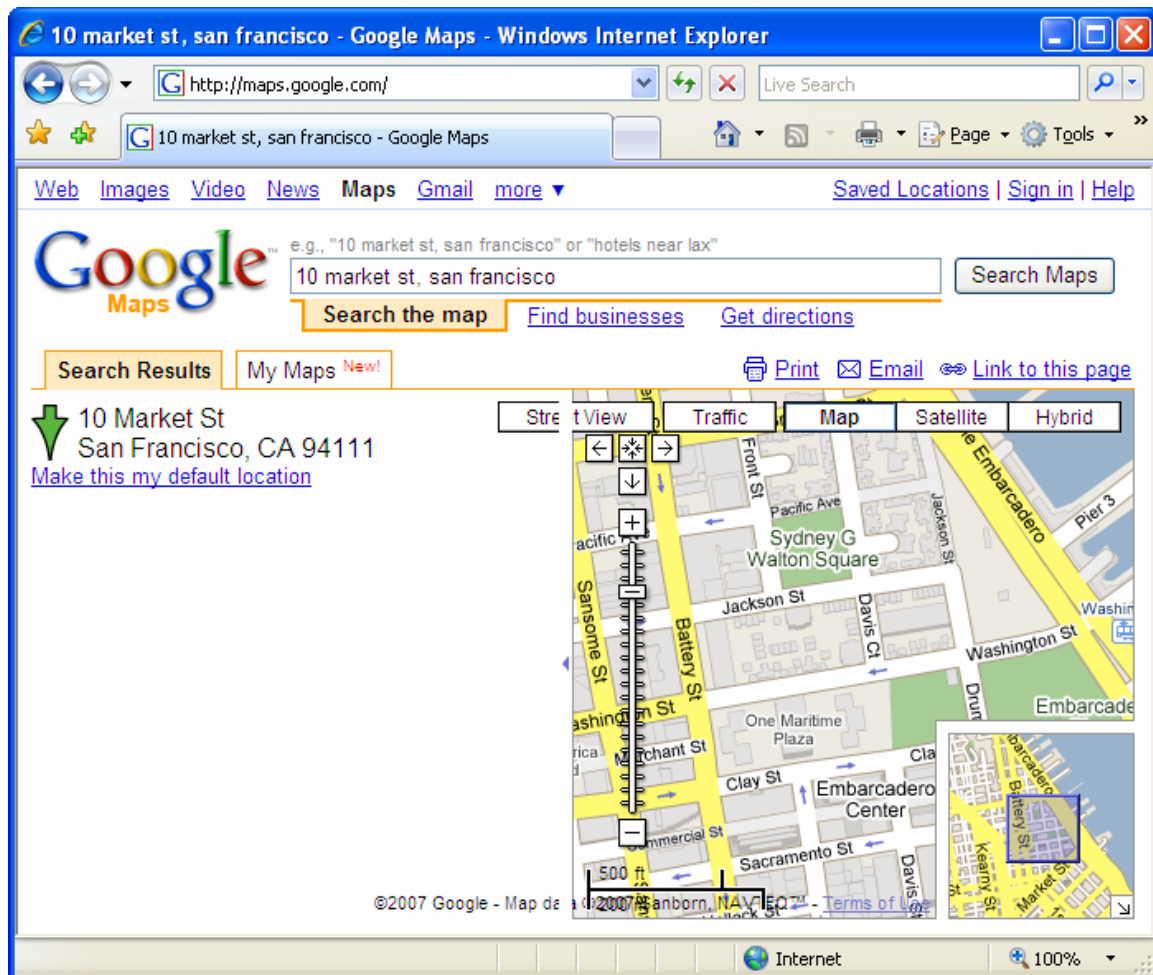
# AJAX

---

- **Although remote scripting has been around for a long time, it became popular in 2005 with the introduction of the acronym *AJAX*.**
  - Originally introduced in the Java community, AJAX is now popular in .NET, PHP, and other platforms.
- **AJAX stands for *Asynchronous JavaScript and XML*.**
  - The asynchronous nature of the request to the server means that the user can continue to interact with the browser-based application while waiting for the data.
  - Despite the XML in the name, data sent to and from the server does not have to be XML documents.
- **Remote scripting enables out-of-band HTTP requests.**
  - The normal mechanism for submitting HTTP requests results in the return of an entire page.
  - An out-of-band request can be tailored to what is most suitable for the task at hand, and will normally call for much less data to be sent.
  - An AJAX framework provides a **proxy** that can be called from JavaScript code to issue the out-of-band HTTP requests.
- **The other key ingredient that makes the technology work is an updatable DOM.**
  - Client code does a partial update of the page.

# Google Maps

- A killer application for AJAX was Google maps.
  - The client-side functionality has tremendous performance, zooming in and out, moving the map about, and so on.
  - Out-of-band HTTP requests bring back only the needed data.



# ASP.NET AJAX

---

- **Although it is quite feasible to create AJAX-enabled Web applications without any special tools, it is much easier with an AJAX framework.**
  - ASP.NET AJAX is a free framework from Microsoft that greatly simplifies creating AJAX applications using ASP.NET and Visual Studio 2008.
  - It is bundled into the .NET 3.5 release of ASP.NET with Visual Studio 2008, and it is also available as a separate download for ASP.NET 2.0.
  - ASP.NET AJAX provides both server-side and client-side services, as well as integration with Visual Studio 2008.
- **Some of the key features include:**
  - Performing significant parts of a Web page's processing in the browser
  - Controls to simplify AJAX programming
  - Partial-page updates
  - Client integration with forms authentication and user profiles.
  - Calls to Web Services from client script
- **Visual Studio enhancements include:**
  - A new group AJAX Extensions in the Toolbox, including controls such as ScriptManager and UpdatePanel.

# Partial Page Rendering

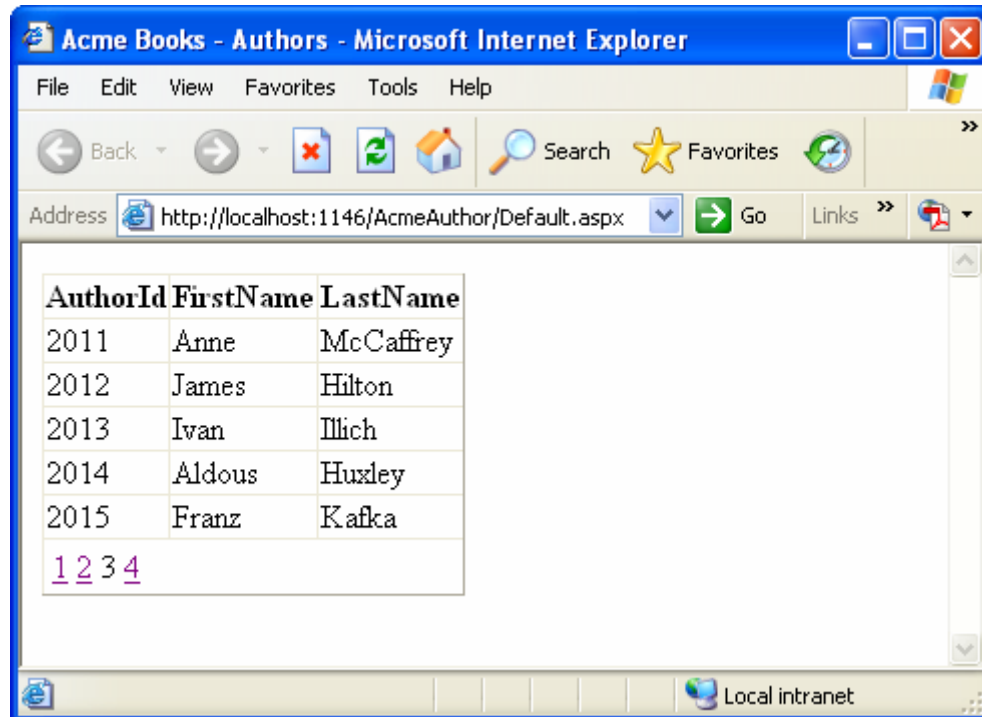
---

- **The normal behavior of a Web application is to display the entire page that has been fetched from a Web server (this is called Full Page Refresh).**
  - This typically results in a visible flicker when the page is viewed in a browser.
- **AJAX provides a model of partial page rendering, where only individual regions of a page that have been changed are updated after a postback.**
- **Thus the whole page does not reload, making user interaction more seamless, removing the flicker.**
- **With ASP.NET AJAX you can implement partial page rendering without having to write custom JavaScript code.**
- **The programming model is familiar to ASP.NET programmers, involving working with Web Server controls and server-side coding.**
  - Client-side scripts can enhance the application, but much can be done without having to write JavaScript manually.

# Partial Page Rendering Example

---

- **Run *AcmeAuthor* in the chapter folder.**
  - This program accesses the database **AcmeBook.mdf** in the **OIC\Data** folder.



- **You may page through the Author table without incurring a full postback and resulting flicker.**



# UpdatePanel Control

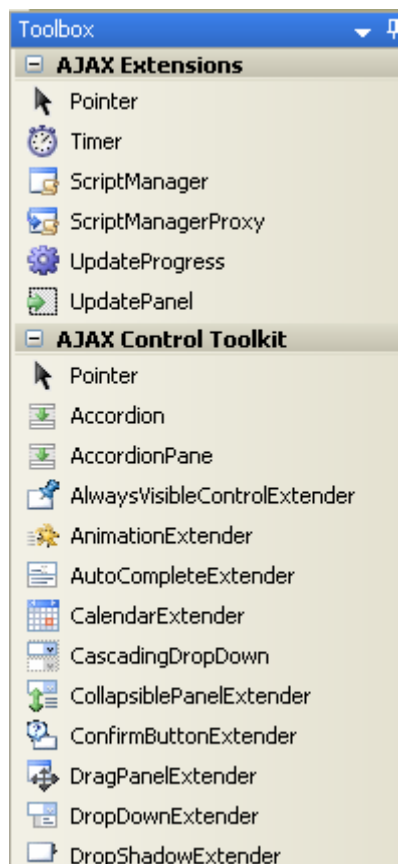
---

- **ASP.NET AJAX makes it very easy to implement partial page rendering in ASP.NET Web apps.**
- **An *UpdatePanel* control specifies a region of a Web page that can be updated without refreshing the whole page.**
- **You must also have a *ScriptManager* control on any page that supports partial page rendering.**
  - The ScriptManager control is also required to use the Microsoft AJAX Client library.
- **Using UpdatePanel is very easy:**
  - In Visual Studio create a new ASP.NET Web site. Add a ScriptManager control.
  - Drag an UpdatePanel control onto your page.
  - Then drag other standard Web server controls you wish to participate in the partial page update into the ContentTemplate area of the UpdatePanel control.

# AJAX Extensions Controls

---

- **ASP.NET AJAX brings a new group of controls in Visual Studio, AJAX Extensions.**
- **Installing the AJAX Control Toolkit creates a much larger set of new controls.**
  - We'll discuss the AJAX Control Toolkit later in the chapter.

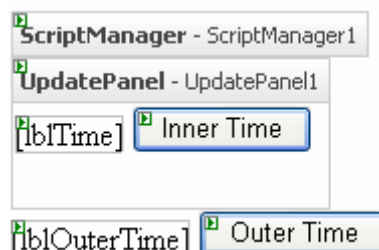


# UpdatePanel Demo

---

- **We can illustrate the use of UpdatePanel with a simple demo.**
1. In the **Demos** folder create a new ASP.NET Web Site **SimpleUpdatePanel**.
  2. Drag a ScriptManger control from the AJAX Extensions group onto your page.
  3. Drag an UpdatePanel control from the AJAX Extensions group onto your page.
  4. Drag a label control and a button inside the UpdatePanel.
  5. Drag another label and button outside the UpdatePanel. Assign the ID and Text properties as shown:

(ID)	Text
lblTime	(blank)
btnInnerTime	Inner Time
lblOuterTime	(blank)
btnOuterTime	Outer Time



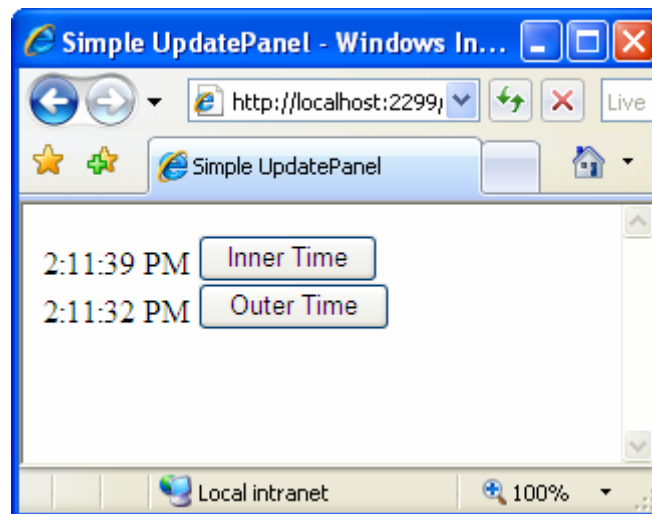
## UpdatePanel Demo (Cont'd)

---

6. Provide the following code in **Page\_Load()**.

```
protected void Page_Load(object sender,
EventArgs e)
{
    string strTime =
        DateTime.Now.ToLongTimeString();
    lblTime.Text = strTime;
    lblOuterTime.Text = strTime;
}
```

7. Run the program and click the Inner Time button a few times.



8. Observe that clicking Inner Time does not cause a full postback, and only the label inside the UpdatePanel is updated.
9. Click the Outer Time button. Now there is a full postback. You will probably see a flicker and a green progress bar in the status panel of the browser. The whole page is refreshed. The two labels will now show the same time.

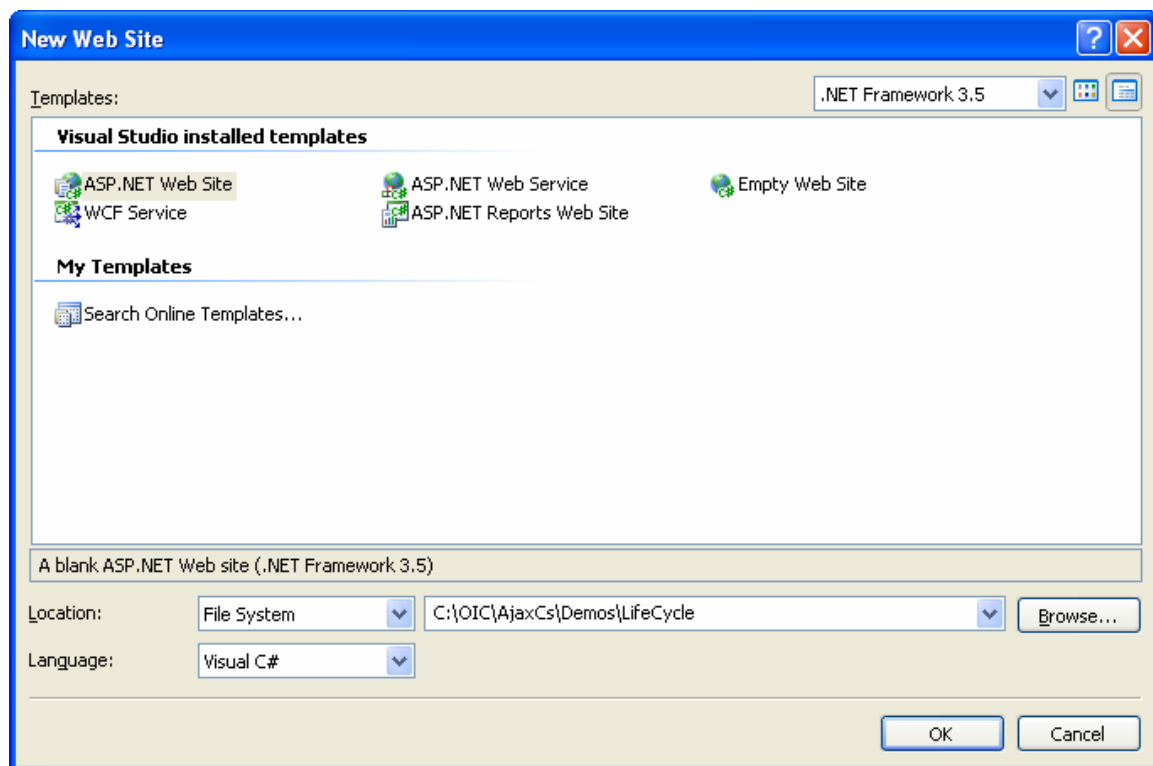
# AJAX Client Library

---

- **The AJAX Client Library consists of JavaScript .js files that extend the capabilities of JavaScript and enable communication with server-side components.**
- **Core services extend the capabilities of JavaScript, including:**
  - Extensions of JavaScript objects
  - A type system supporting object-oriented features such as namespaces, classes and inheritance
  - Support for events and serialization
- **A networking layer supports asynchronous requests and calling Web services.**
- **A browser compatibility layer provides compatibility across frequently used browsers such as Internet Explorer, Firefox and Safari.**
- **Component classes support the creation of client-side controls and non-visual components.**

# Using the Client Library

- **To make use of the client library you must create an ASP.NET AJAX enabled Web site.**
  - You may use either Visual Studio 2008 or Visual Web Developer 2008 Express Edition.
  - First, create an ASP.NET Web Site.



- Next, switch to Design view and drag a ScriptManager control from the AJAX Extensions group onto the **Default.aspx** page.

# ScriptManager Control

---

- **Your new Web site now has a *ScriptManager* control in its *Default.aspx* page.**

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default" %>

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

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ScriptManager ID="ScriptManager1"
                                runat="server">
            </asp:ScriptManager>
        </div>
    </form>
</body>
</html>
```

- **The *ScriptManager* control manages client script for pages in ASP.NET AJAX applications.**
  - It registers the script of the .js files of the AJAX Client Library for use in the page.
  - This gives you access to the features of the AJAX Client Library.

# Embedded JavaScript Files

---

- **The .js files of the client library are provided as embedded resources in the assembly *System.Web.Extensions*.**
  - Access to this assembly is provided in the **Web.config** file.

```
<assemblies>
    ...
    <add assembly="System.Web.Extensions,
        Version=3.5.0.0, Culture=neutral,
        PublicKeyToken=31BF3856AD364E35" />
    ...
</assemblies>
```

- **You may also place your own .js files as embedded resources in an assembly, making your scripts easier to deploy.**
  - Normally such a resource assembly will be placed in the **Bin** folder of your Web application.
  - We will see how to do this later in the chapter.



# Client Library Namespaces

---

- **The client library functionality is provided in a number of namespaces.**
  - Do not confuse these client-side namespaces, rooted in **Sys**, with server-side namespaces rooted in **System**.
- **The client-side namespaces include:**
  - **Global namespace** which contains extensions to familiar JavaScript objects such as **Array**, **String**, **Object**, **Number**, **Error**, and so on. It also contains functions such as **\$get()** that simplify working with the DOM.
  - **Sys** is the root namespace for the library containing fundamental classes and base classes.
  - **Sys.Net** provides support for communication between client and server code.
  - **Sys.Serialization** provides support for data serialization in client code.
  - **Sys.Services** supports client access to application services such as authentication and profile.
  - **Sys.UI** provides client-side UI elements such as controls and events.
  - **Sys.WebForms** supports partial-page updating in ASP.NET AJAX client pages.

# Sys.Debug Tracing

---

- **The *Debug* class in the *Sys* namespace provides client-side debugging support.**
  - It is analogous to the **Debug** class in the **System.Diagnostics** namespace on the server.
- **The *trace()* method displays text to the debugger console.**

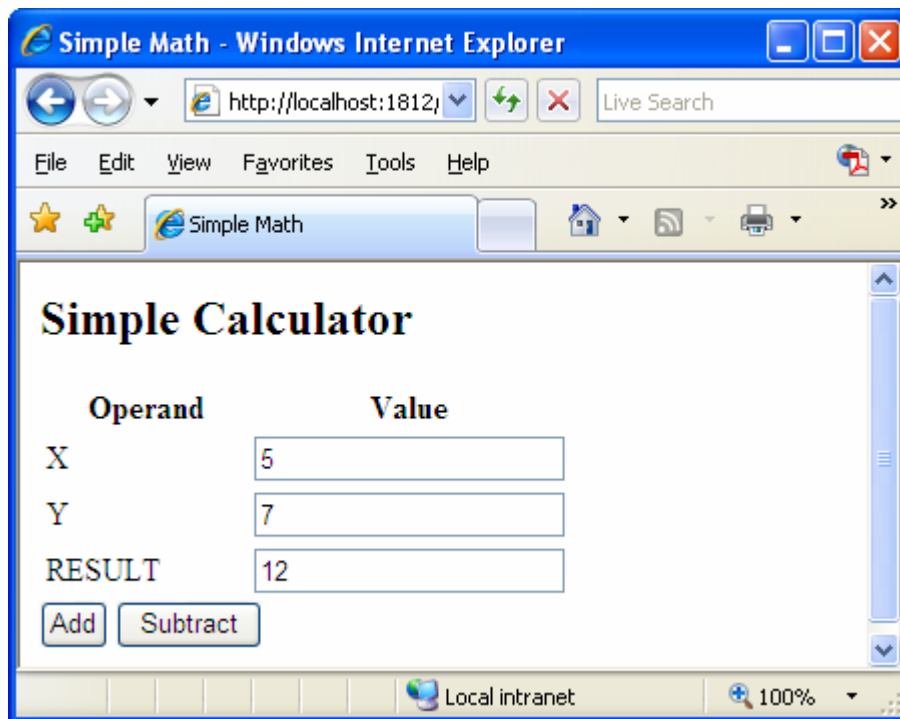
```
Sys.Debug.trace("Sample trace output");
```

- You can view the debugger console in Visual Studio in the Output window.
- **Note that in JavaScript you must always use the fully-qualified names of your classes.**
  - There is no analogue of the C# **using** or Visual Basic **Imports** statement.

# Simple Client Library Example

---

- Let's see how the Client Library can simplify the coding of client-side scripts.
- Our example program does simple arithmetic on integers using client-side JavaScript.



- See **SimpleMath** in the chapter folder.
- Step 1 uses JavaScript and the DOM without use of the Client Library.
- Step 2 shortens our code with use of the Client Library.
- Step 3 makes our script disappear entirely from our Web page by embedding it as a resource in a binary file.

# Document Object Model

---

- The strict rules of XHTML ensure that the document has a hierarchical structure, enabling the Document Object Model (DOM) to work correctly.
- The DOM provides a browser-independent way of representing a document.
- The user can access the document via a common set of objects, properties, methods, and events.
- The user can also alter the contents of a Web page dynamically using scripts.
- An example of a commonly used DOM method is *getElementById()*.
  - This method gets the DOM element that has the specified **id** attribute.

```
var tX = document.getElementById("txtX");
```

# JavaScript for Simple Calculator

---

- **Here is the Step 1 JavaScript code.**

- Note the extensive use of **getElementById()**.

```
function Add()  
{  
    var tX = document.getElementById("txtX");  
    var x = parseInt(tX.value);  
    var tY = document.getElementById("txtY");  
    var y = parseInt(tY.value);  
    var sum = x + y;  
    var tSum = document.getElementById("txtResult");  
    tSum.innerHTML = sum;  
}  
function Subtract()  
{  
    var tX = document.getElementById("txtX");  
    var x = parseInt(tX.value);  
    var tY = document.getElementById("txtY");  
    var y = parseInt(tY.value);  
    var diff = x - y;  
    var tDiff =  
        document.getElementById("txtResult");  
    tDiff.innerHTML = diff;  
}
```

- **Here is the XHTML that hooks the JavaScript functions to the buttons.**

```
<asp:Button ID="btnAdd" runat="server" Text="Add"  
    OnClientClick="Add()" />  
<asp:Button ID="btnSubtract" runat="server"  
    Text="Subtract" OnClientClick="Subtract()" />
```

## Using the Client Library

---

- **Step 2 uses the global library function *\$get()* as a shortcut for *getElementById()*.**
  - It also uses the **trace()** method of the **Debug** class in the **Sys** namespace.
  - This output appears in the Output window of Visual Studio when run under the debugger.

```
function Add()  
{  
    Sys.Debug.trace("Add");  
    var x = parseInt($get("txtX").value);  
    var y = parseInt($get("txtY").value);  
    var sum = x + y;  
    $get("txtResult").innerText = sum;  
}  
function Subtract()  
{  
    Sys.Debug.trace("Subtract");  
    var x = parseInt($get("txtX").value);  
    var y = parseInt($get("txtY").value);  
    var diff = x - y;  
    $get("txtResult").innerText = diff;  
}
```

- **To access the client library, a **ScriptManager** control must be included in the page.**

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

# JavaScript in Assemblies

---

- **An alternative to distributing JavaScript in source .js files is to embed it as a resource in an assembly.**
  - This assembly can then be provided in the Bin folder of the ASP.NET Web application.
- **The steps to do this are quite simple.**
  1. Create a class library project. (You can do this in Visual Studio or Visual C# Express Edition, but not in Visual Web Developer.)
  2. Provide references to **System.Web** and **System.Web.Extensions**.
  3. Add the .js file(s) to be embedded to your project. (If you used inline JavaScript, put this code into one or more .js files.)
  4. In the Properties window of each .js file, set Build Action to Embedded Resource.
  5. In the **AssemblyInfo.cs** file (under Properties in Solution Explorer) add this line:

```
[assembly: System.Web.UI.WebResource(
    "<AssemblyName>.<JavaScriptFile>.js",
    "application/x-javascript" ) ]
```
  6. Build the class library assembly.
  7. Copy the assembly to the Bin folder of your Web application.

## Providing a ScriptReference

---

8. Edit the **Default.aspx** file in your Web application to provide a **ScriptReference** to the assembly. This is in a `<Scripts>` tag inside the `<asp:ScriptManager>` tag.

```
<asp:ScriptManager ID="ScriptManager1"
runat="server" >
  <Scripts>
    <asp:ScriptReference Assembly="<Assembly>"
      Name="<Assembly>.<JavaScriptFile>.js" />
  </Scripts>
</asp:ScriptManager>
```

9. In **Default.aspx** edit the `<script>` tag so that it references the JavaScript file rather than uses inline JavaScript code.

```
<script type="text/javascript"
      src="<JavaScriptFile>.js">
</script>
```

- **Step 3 of the *SimpleMath* Web application illustrates use of JavaScript embedded in an assembly.**
  - The assembly containing the embedded JavaScript is created in the class library project in **SimpleMathLib** in the chapter directory.



# AJAX Control Toolkit

---

- **While there are only a few new controls provided with ASP.NET AJAX itself, there is a rich collection of controls available in the *AJAX Control Toolkit* or *ACT*.**
- **The ACT is the fruit of a joint open-source project between Microsoft and the ASP.NET community to provide Web-client components that work with ASP.NET AJAX.**
  - Information about ACT and download is available from Codeplex:

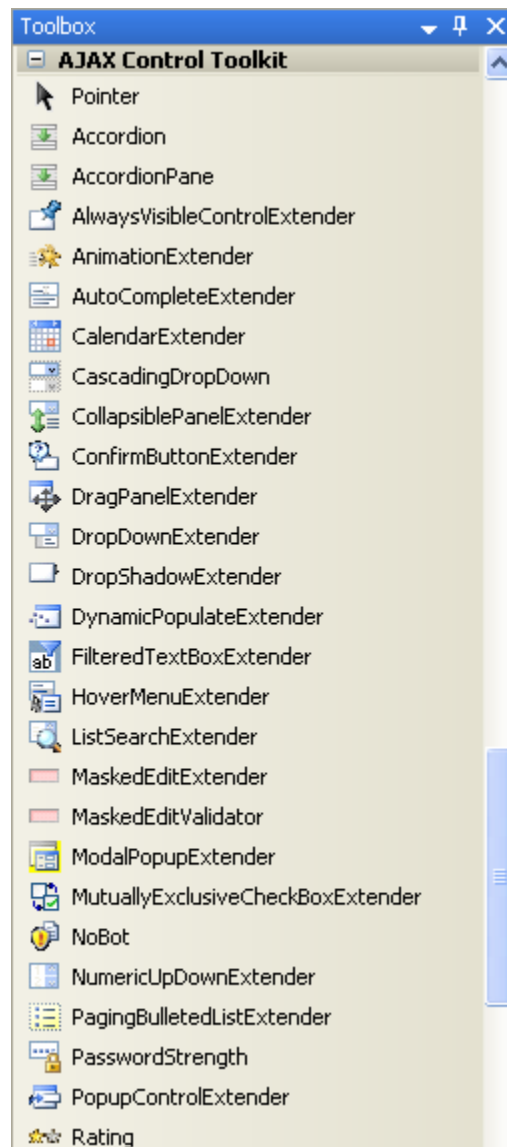
<http://www.codeplex.com/AjaxControlToolkit>

- **ACT is available in two ways:**
  - A complete assembly that can be registered with any application where you use ACT.
  - Source code that can be selectively incorporated into your application.
- **In this chapter we'll use the complete assembly.**

# ACT Controls in Visual Studio

---

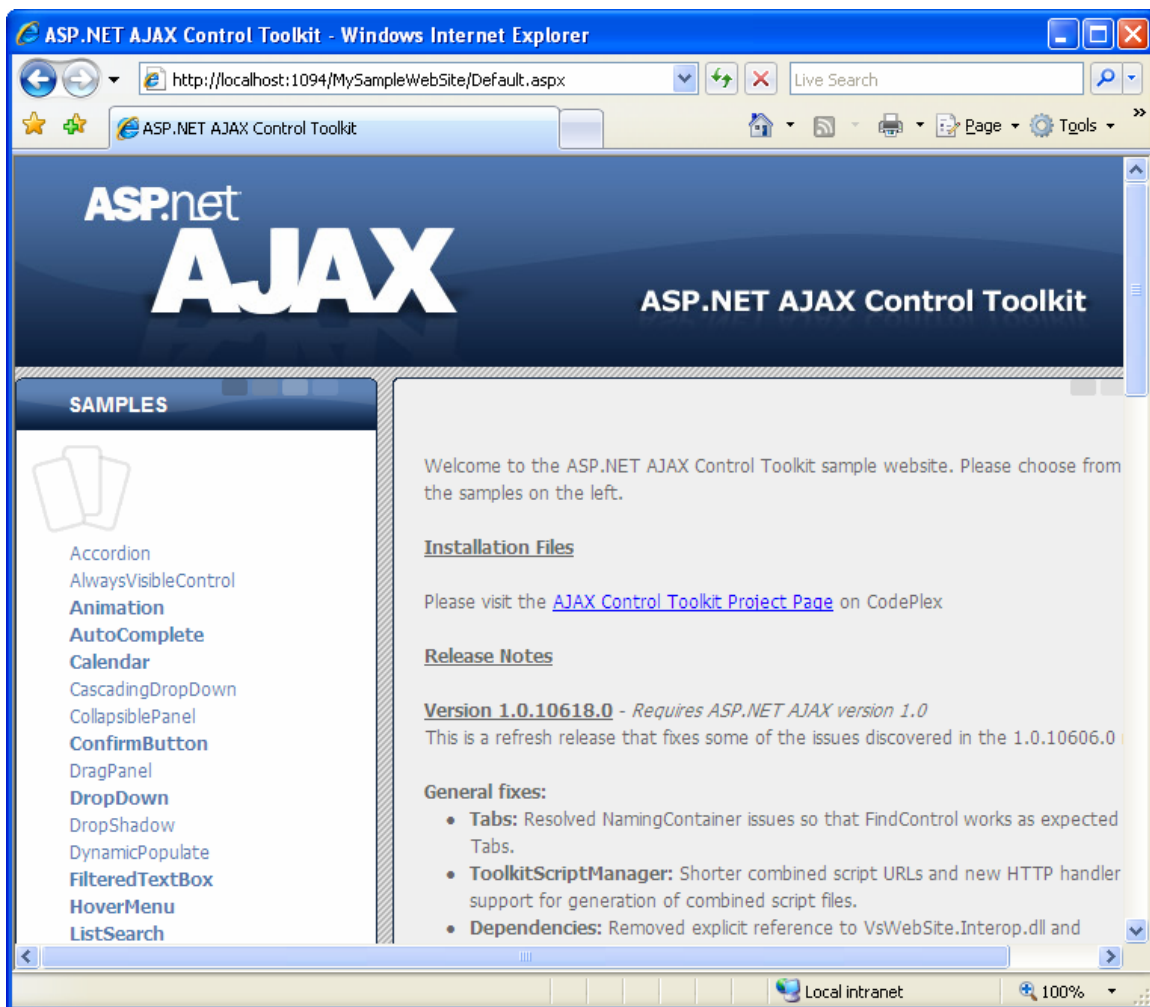
- **Once ACT is installed, you will have a new toolbox group in Visual Studio with a great many controls.**



# ACT Sample Web Site

---

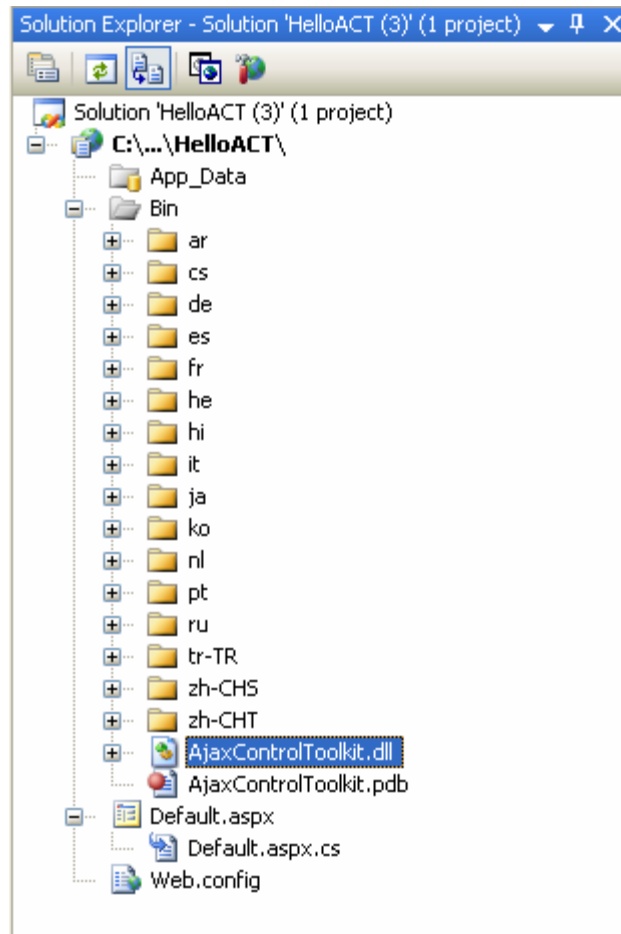
- **ACT provides a sample Web site, which you can open and run in Visual Studio.**
  - Open and run the solution **AjaxControlToolkit.sln**.
  - There is documentation and sample code for every control.



# AjaxControlToolkit.dll

---

- When you add an ACT control to your project, Visual Studio will automatically create a Bin folder and populate it with the *AjaxControlToolkit.dll* assembly.
- There will be folders for versions of this assembly for multiple languages.



- You may delete these extra folders if you don't care about international issues.
- The only essential file is **AjaxControlToolkit.dll**.

# Registering AjaxControlToolkit.dll

---

- **You must register *AjaxControlToolkit.dll* on every page that uses an ACT control.**

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

- **This code is inserted automatically into your page when you drag an ACT control from the toolbox onto your page.**
- **The TagPrefix is arbitrary, but must be used in the markup for the control.**

```
<cc1:NumericUpDownExtender
    ID="NumericUpDownExtender1" runat="server"
    TargetControlID="txtRating"
    Minimum="1"
    Maximum="7"
    Width="60"
    >
</cc1:NumericUpDownExtender>
```

## Extender Controls

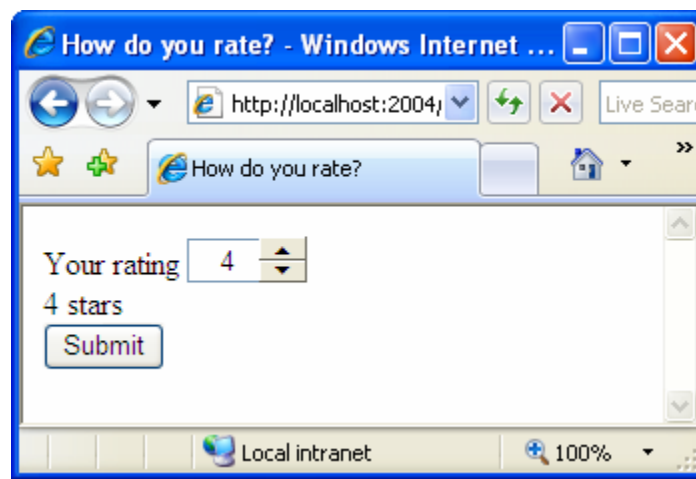
---

- An *extender control* is a server control that represents logical behavior that can be attached to one or more control types to extend their behavior.
- **ASP.NET AJAX** does not provide any concrete extender controls itself, but it provides a base class *ExtenderControl*.
  - The class defines the property **TargetControlID**, which is a string representing the ID of the server control being extended.
  - Every extender control must provide this property.
- **The AJAX Control Toolkit** provides a number of off-the-shelf extender controls.
- **The ACT** also provides classes and a template to enable you to roll your own extender controls.

# NumericUpDownExtender Control

---

- An example of an extender control is *NumericUpDown*.
  - This control extends TextBox.
  - It provides “up” and “down” buttons that increment and decrement the value in the TextBox.
- For an example, see *NumericUpDown* in the chapter directory.



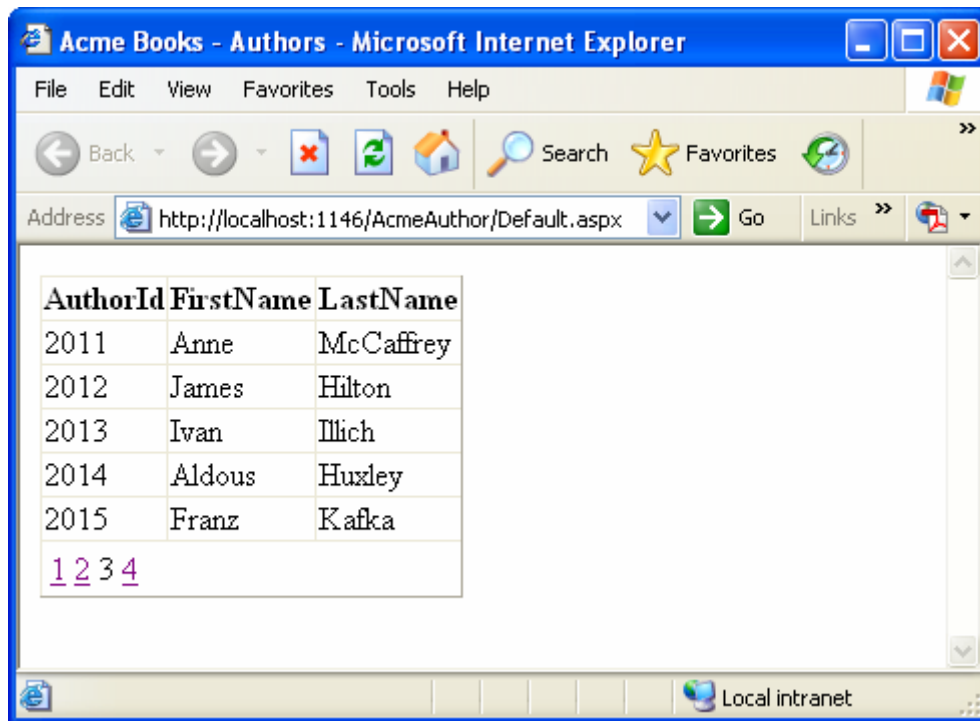
```
<asp:TextBox ID="txtRating"
             runat="server">4</asp:TextBox>
...
<ccl:NumericUpDownExtender
  ID="NumericUpDownExtender1" runat="server"
  TargetControlID="txtRating"
  Minimum="1"
  Maximum="7"
  Width="60"
  >
</ccl:NumericUpDownExtender>
```

- Width includes both the target control and extender control.

# Lab 14

## An Acme Database Application

In this lab you will implement a Web application that displays authors in the AcmeBook database. By employing AJAX you will provide an effective, flicker-free user interface.



Detailed instructions are contained in the Lab 14 write-up at the end of the chapter.

Suggested time: 20 minutes.



# Summary

---

- **Rich Internet applications (RIAs) can provide some of the best features of desktop and Web applications.**
- **Plug-ins such as Flash and Microsoft's new Silverlight are one approach to implementing RIAs, but they require running substantial code on the client.**
- **JavaScript, implemented in all modern browsers, is a lightweight approach to providing client-side functionality.**
- **Asynchronous communication with the server can provide a more responsive user interface.**
- **AJAX, or Asynchronous JavaScript and XML, is a term for technologies enabling highly interactive Web applications that give the user a better experience.**
- **Partial page rendering in AJAX applications can enhance the user's experience, and you can implement it in ASP.NET AJAX applications with the ScriptManager and UpdatePanel control.**
- **The AJAX Client Library provides a framework that simplifies client-side programming.**
- **You can use the AJAX Control Toolkit (ACT) with Visual Studio to create Web applications with rich user interfaces.**

## Lab 14

### An Acme Database Application

#### Introduction

In this lab you will implement a Web application that displays authors in the AcmeBook database. By employing AJAX you will provide an effective, flicker-free user interface.

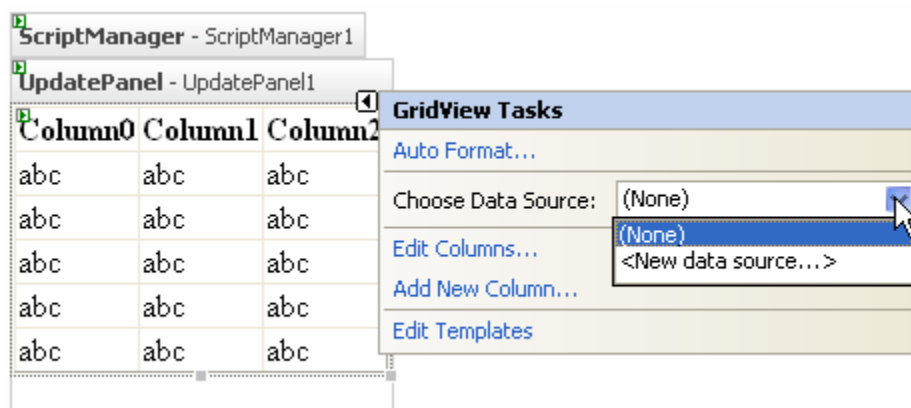
**Suggested Time:** 20 minutes.

**Root Directory:** OIC\AspCs

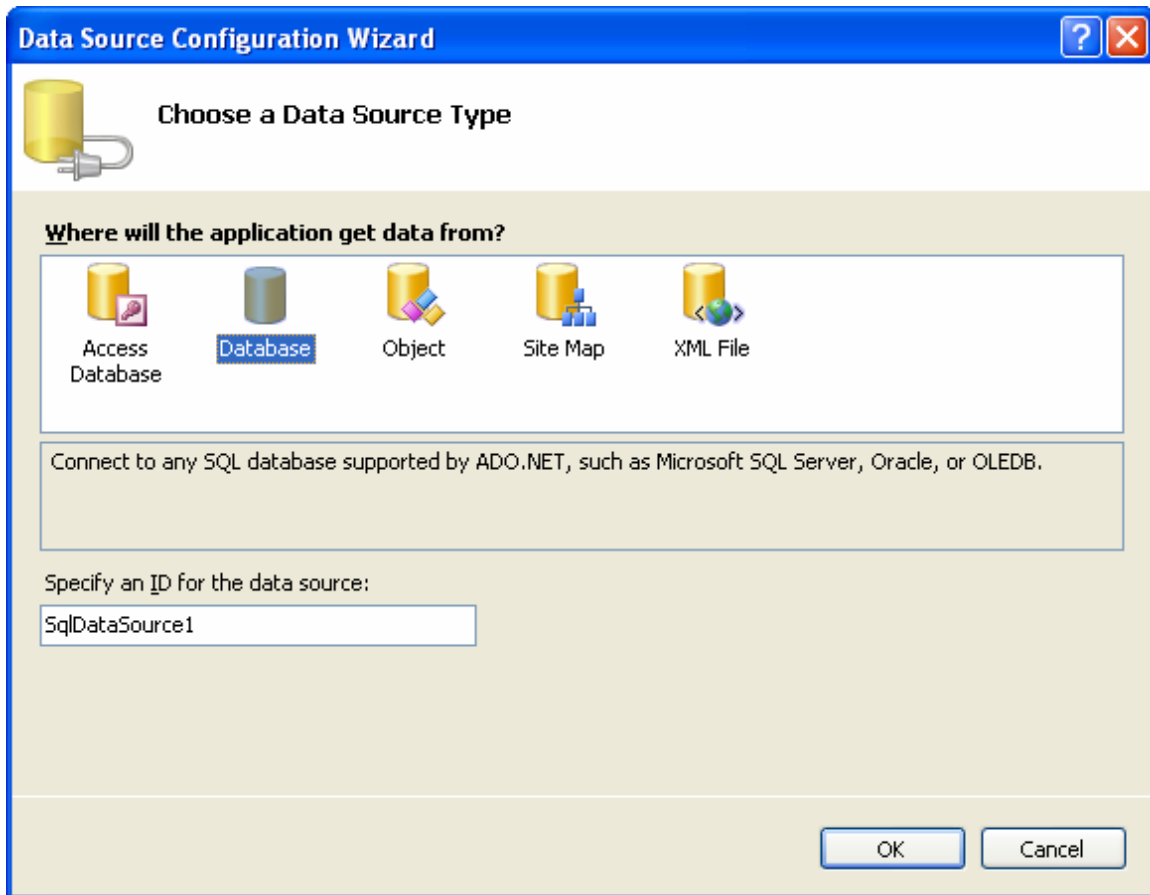
**Directories:** Labs\Lab14\AcmeAuthor (do your work here)  
Chap14\AcmeAuthor (solution)

#### Instructions

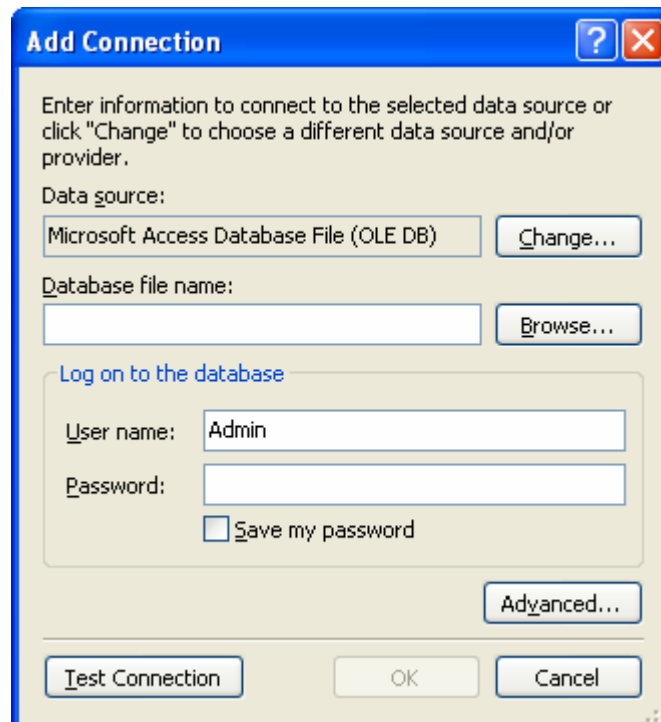
1. Create a new AJAX enabled Web site AcmeAuthor in the Lab14 folder. Remember that you'll need to drag a ScriptManager control onto your page. Make the title of your default web page "Acme Books – Authors." Make sure that you are in the default "flow" layout mode. (In Tools | Options, select HTML Designer and CSS Styling, and ensure that the "Change positioning to absolute ..." checkbox is cleared.)
2. Drag an UpdatePanel onto your page. Inside this panel, place a GridView and name it gvAuthor.
3. In the GridView tasks, drop down the list in Choose Data Source.



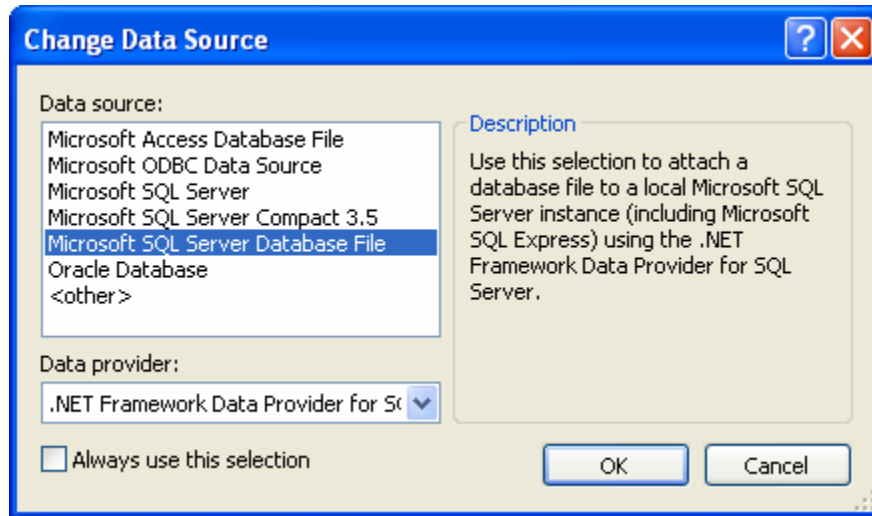
4. Select <New data source...>.
5. The DataSource Configuration wizard comes up. Select Database.



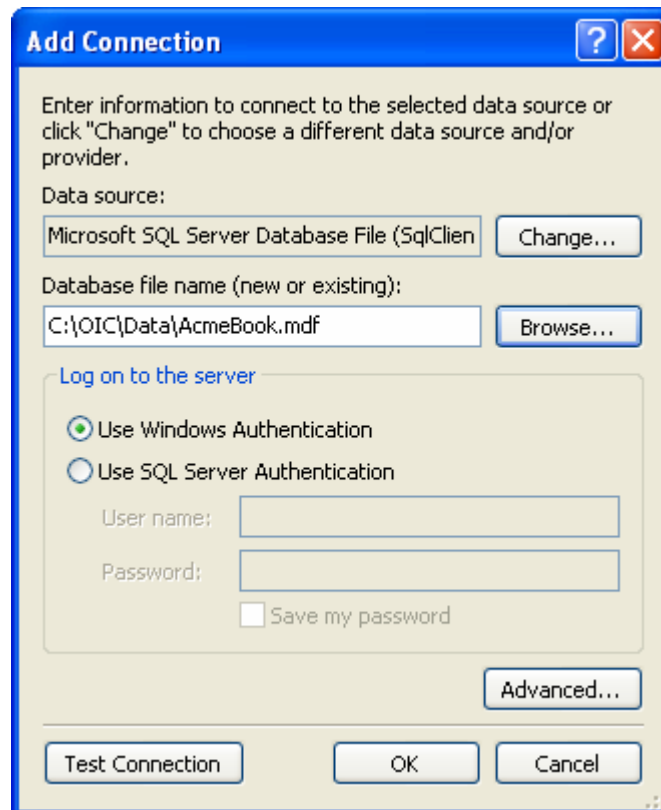
6. Click OK. The Add Connection dialog comes up (if not, click "New Connection").



- Click Change and select Microsoft SQL Server Database File as the data source.

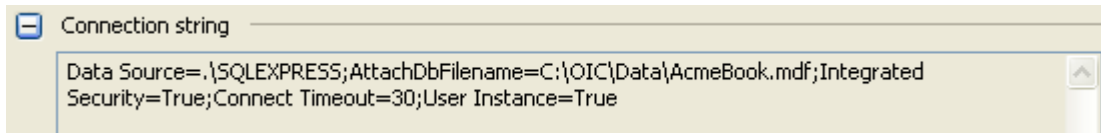


- Click OK. The Add Connection dialog comes up. Browse to the file **AcmeBook.mdf** in **OIC\Data**.



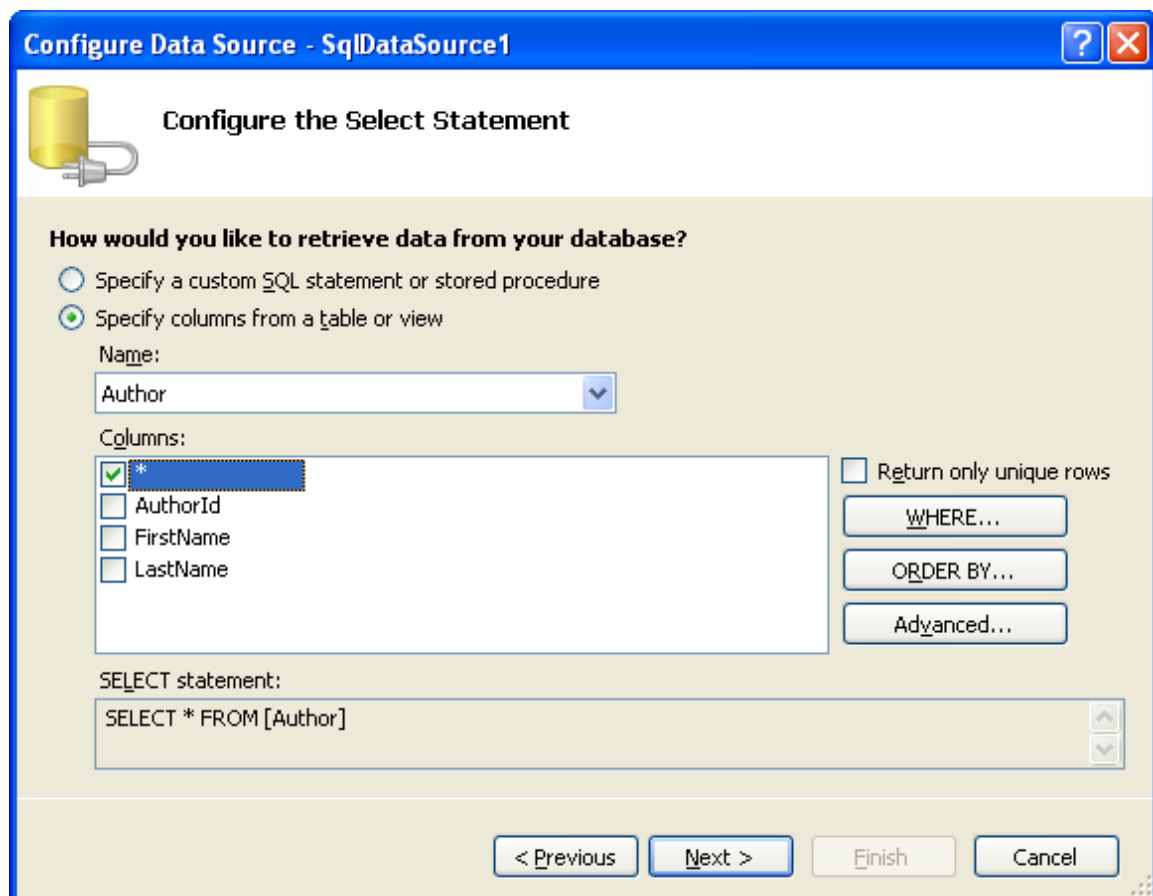
- Click the Test Connection button at the bottom of the page to make sure you've entered the correct connection information. Click OK.

10. Back in the Configure Data Source wizard you can now examine the connection string by clicking the little plus sign to expand it.



11. Click Next. You will be asked if you want to save the connection string in the application configuration file. Accept the suggestion of yes by clicking Next.

12. In “Configure the Select Statement” choose the Author table and \* to choose all columns.

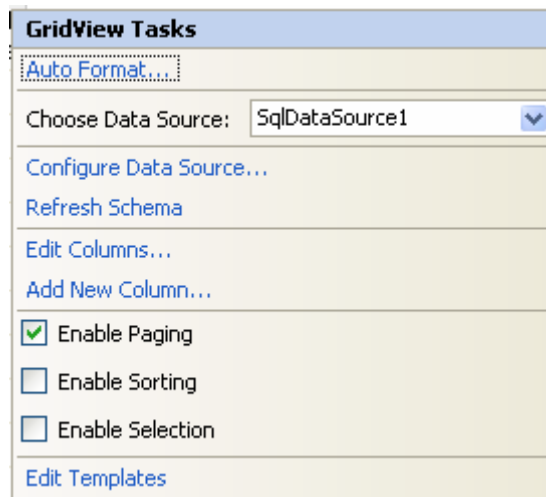


13. Click Next. In the following step you can test the query.

AuthorId	FirstName	LastName
2001	Robert	Oberg
2002	Dana	Wyatt
2003	Michael	Stiefel

14. Click Finish.

15. Back in the GridView tasks check Enable Paging.



16. Build and run the application. You will see two pages of data, and you can switch between the pages without seeing any flicker.

17. Change the PageSize property of the GridView to 5. Build and run. You are done!

