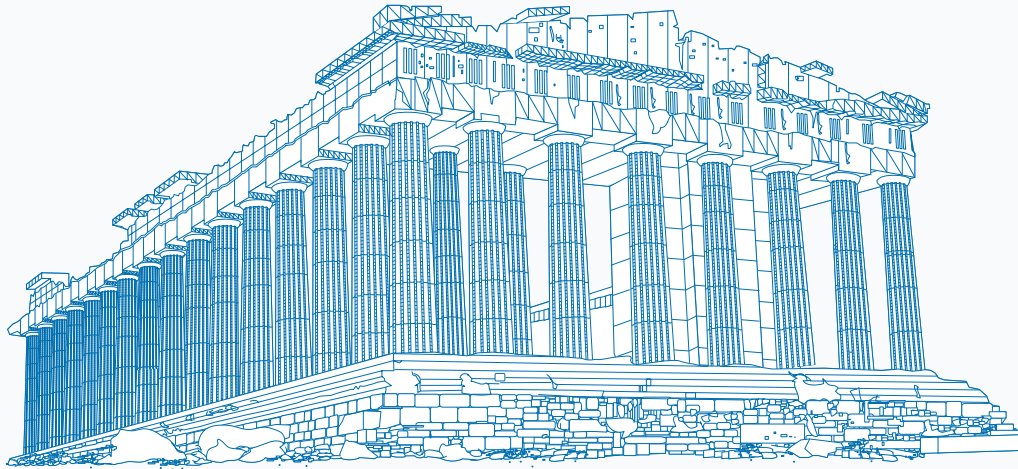


---

# ASP.NET

*Your visual blueprint for  
creating Web applications on the .NET framework  
by Danny Ryan and Tommy Ryan*



From

**maranGraphics®**

&



**Hungry Minds™**

Best-Selling Books • Digital Downloads • e-Books • Answer Networks • e-Newsletters • Branded Web Sites • e-Learning

New York, NY • Cleveland, OH • Indianapolis, IN

---

---

## ASP.NET: Your visual blueprint for creating Web applications on the .NET framework

---

Published by  
Hungry Minds, Inc.  
909 Third Avenue  
New York, NY 10022

Copyright © 2002 Hungry Minds, Inc.

Certain *designs/text/illustrations* copyright © 1992–2001 maranGraphics, Inc., used with maranGraphics' permission. All rights reserved. No part of this book, including interior design, cover design, and icons, may be reproduced or transmitted in any form, by any means (electronic, photocopying, recording, or otherwise) without the prior written permission of the publisher.

maranGraphics, Inc.  
5755 Coopers Avenue  
Mississauga, Ontario, Canada  
L4Z 1R9

Library of Congress Control Number: 2001090695

ISBN: 0-7645-3617-6

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

1V/RX/RQ/QR/IN

Distributed in the United States by Hungry Minds, Inc.

Distributed by CDG Books Canada Inc. for Canada; by Transworld Publishers Limited in the United Kingdom; by IDG Norge Books for Norway; by IDG Sweden Books for Sweden; by IDG Books Australia Publishing Corporation Pty. Ltd. for Australia and New Zealand; by TransQuest Publishers Pte Ltd. for Singapore, Malaysia, Thailand, Indonesia, and Hong Kong; by Gotop Information Inc. for Taiwan; by ICG Muse, Inc. for Japan; by Intersoft for South Africa; by Eyrolles for France; by International Thomson Publishing for Germany, Austria and Switzerland; by Distribuidora Cuspide for Argentina; by LR International for Brazil; by Galileo Libros for Chile; by Ediciones ZETA S.C.R. Ltda. for Peru; by WS Computer Publishing Corporation, Inc., for the Philippines; by Contemporanea de Ediciones for Venezuela; by Express Computer Distributors for the Caribbean and West Indies; by Micronesia Media Distributor, Inc. for Micronesia; by Chips Computadoras S.A. de C.V. for Mexico; by Editorial Norma de Panama S.A. for Panama; by American Bookshops for Finland.

For U.S. corporate orders, please call maranGraphics at 800-469-6616 or fax 905-890-9434.

For general information on Hungry Minds' products and services please contact our Customer Care Department within the U.S. at 800-762-2974, outside the U.S. at 317-572-3993 or fax 317-572-4002.

For sales inquiries and reseller information, including discounts, premium and bulk quantity sales, and foreign-language translations, please contact our Customer Care Department at 800-434-3422, fax 317-572-4002, or write to Hungry Minds, Inc., Attn: Customer Care Department, 10475 Crosspoint Boulevard, Indianapolis, IN 46256.

For information on licensing foreign or domestic rights, please contact our Sub-Rights Customer Care Department at 212-884-5000.

For information on using Hungry Minds' products and services in the classroom or for ordering examination copies, please contact our Educational Sales Department at 800-434-2086 or fax 317-572-4005.

For press review copies, author interviews, or other publicity information, please contact our Public Relations department at 317-572-3168 or fax 317-572-4168.

For authorization to photocopy items for corporate, personal, or educational use, please contact Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, or fax 978-750-4470.

Screen shots displayed in this book are based on pre-released software and are subject to change.

---

## Trademark Acknowledgments

---

Hungry Minds, the Hungry Minds logo, Visual, the Visual logo, Read Less - Learn More and related trade dress are registered trademarks or trademarks of Hungry Minds, Inc., in the United States and/or other countries and may not be used without written permission. The maranGraphics logo is a registered trademark or trademark of maranGraphics, Inc. Visual Studio is a registered trademark or trademark of Microsoft Corporation. All other trademarks are the property of their respective owners. Hungry Minds, Inc. and maranGraphics, Inc. are not associated with any product or vendor mentioned in this book.

FOR PURPOSES OF ILLUSTRATING THE CONCEPTS AND TECHNIQUES DESCRIBED IN THIS BOOK, THE AUTHORS HAVE CREATED VARIOUS NAMES, COMPANY NAMES, MAILING, E-MAIL AND INTERNET ADDRESSES, PHONE AND FAX NUMBERS, AND SIMILAR INFORMATION, ALL OF WHICH ARE FICTITIOUS. ANY RESEMBLANCE OF THESE FICTITIOUS NAMES, ADDRESSES, PHONE AND FAX NUMBERS, AND SIMILAR INFORMATION TO ANY ACTUAL PERSON, COMPANY AND/OR ORGANIZATION IS UNINTENTIONAL AND PURELY COINCIDENTAL.

---

## Permissions

---

### maranGraphics

Certain text and illustrations by maranGraphics, Inc., used with maranGraphics' permission.



is a trademark of  
Hungry Minds, Inc.

**LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND AUTHOR HAVE USED THEIR BEST EFFORTS IN PREPARING THIS BOOK. THE PUBLISHER AND AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS BOOK AND SPECIFICALLY DISCLAIM ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. THERE ARE NO WARRANTIES WHICH EXTEND BEYOND THE DESCRIPTIONS CONTAINED IN THIS PARAGRAPH. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES REPRESENTATIVES OR WRITTEN SALES MATERIALS. THE ACCURACY AND COMPLETENESS OF THE INFORMATION PROVIDED HEREIN AND THE OPINIONS STATED HEREIN ARE NOT GUARANTEED OR WARRANTED TO PRODUCE ANY PARTICULAR RESULTS, AND THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY INDIVIDUAL. NEITHER THE PUBLISHER NOR AUTHOR SHALL BE LIABLE FOR ANY LOSS OF PROFIT OR ANY OTHER COMMERCIAL DAMAGES, INCLUDING BUT NOT LIMITED TO SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR OTHER DAMAGES.**

### U.S. Corporate Sales

Contact maranGraphics  
at (800) 469-6616 or  
fax (905) 890-9434.

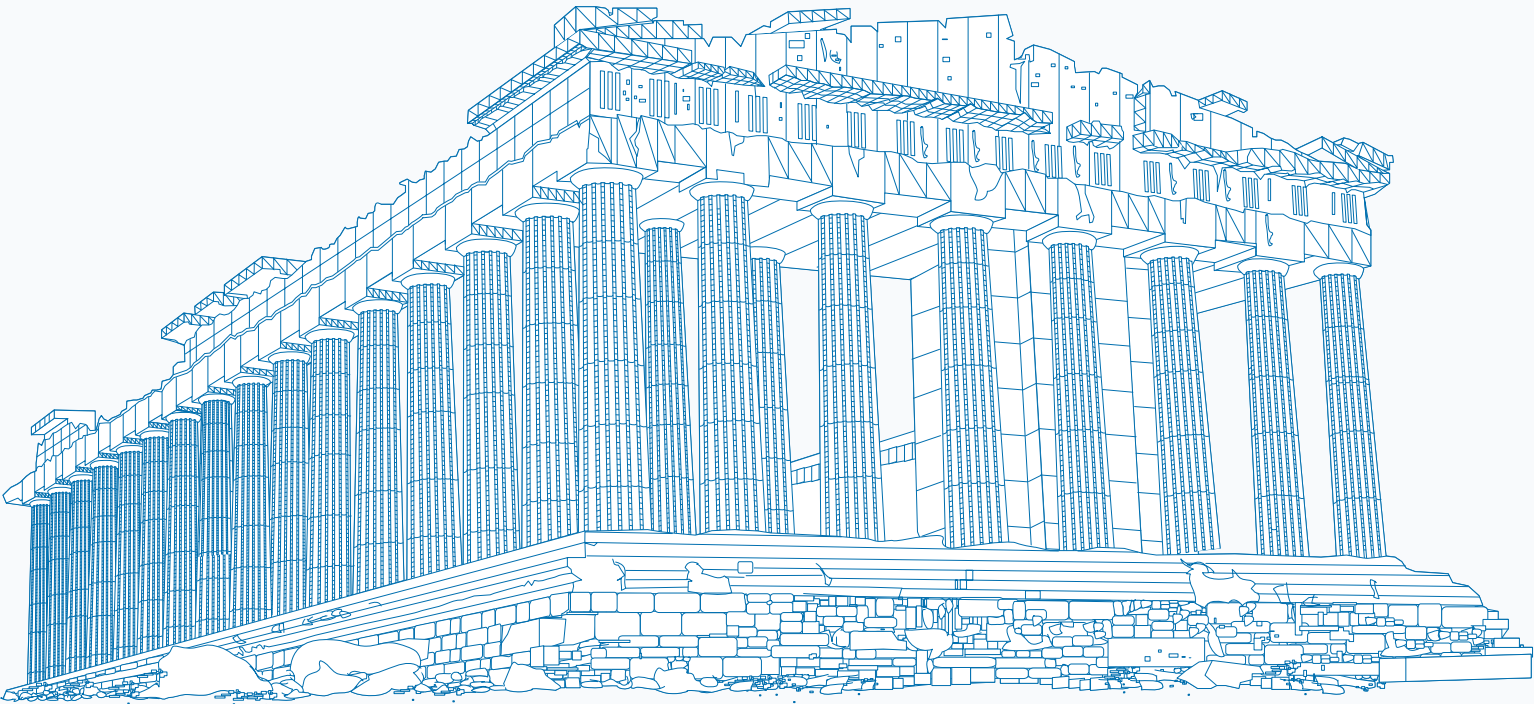
### U.S. Trade Sales

Contact Hungry Minds  
at (800) 434-3422  
or (317) 572-4002.

---

# ASP.NET

*Your visual blueprint for  
creating Web applications on the .NET framework*



---

maranGraphics is a family-run business  
located near Toronto, Canada.



At [maranGraphics](#), we believe in producing great computer books — one book at a time.

maranGraphics has been producing high-technology products for over 25 years, which enables us to offer the computer book community a unique communication process.

Our computer books use an integrated communication process, which is very different from the approach used in other computer books. Each spread is, in essence, a flow chart — the text and screen shots are totally incorporated into the layout of the spread. Introductory text and helpful tips complete the learning experience. maranGraphics' approach encourages the left and right sides of the brain to work together — resulting in faster orientation and greater memory retention.

Above all, we are very proud of the handcrafted nature of our books. Our carefully-chosen writers are experts in their fields, and spend countless hours researching

and organizing the content for each topic. Our artists rebuild every screen shot to provide the best clarity possible, making our screen shots the most precise and easiest to read in the industry. We strive for perfection, and believe that the time spent handcrafting each element results in the best computer books money can buy.

Thank you for purchasing this book. We hope you enjoy it!

Sincerely,

[Robert Maran](#)

President

[maranGraphics](#)

[Rob@maran.com](mailto:Rob@maran.com)

[www.maran.com](http://www.maran.com)

[www.hungryminds.com/visual](http://www.hungryminds.com/visual)

---



---

---

## CREDITS

---

### *Acquisitions, Editorial, and*

#### **Media Development**

##### **Project Editor**

Jade L. Williams

##### **Acquisitions Editor**

Jen Dorsey

##### **Product Development Supervisor**

Lindsay Sandman

##### **Copy Editor**

Timothy Borek

##### **Technical Editor**

John Paul Purdum

##### **Editorial Manager**

Rev Mengle

##### **Media Development Manager**

Laura Carpenter VanWinkle

##### **Permissions Editor**

Carmen Krikorian

##### **Media Development Specialist**

Marisa E. Pearman

##### **Editorial Assistants**

Amanda Foxworth

Candace Nicholson

### **Production**

#### **Book Design**

maranGraphics®

#### **Production Coordinator**

Nancee Reeves

#### **Layout**

LeAndra Johnson, Adam Mancilla,

Kristin Pickett, Jill Piscitelli,

Betty Schulte, Erin Zeltner

#### **Screen Artists**

Ronda David-Burroughs,

David Gregory, Mark Harris,

Jill Proll

#### **Cover Illustration**

Russ Marini

#### **Proofreaders**

Laura L. Bowman,

Andy Hollandbeck, Susan Moritz

Angel Perez, Marianne Santy

#### **Indexer**

TECHBOOKS Production Services

#### **Special Help**

Microsoft Corporation,

Diana Conover, Richard Graves,

Leslie Kersey

---

---

## ACKNOWLEDGMENTS

---

**Hungry Minds, Inc.:** John Kilcullen, CEO; Bill Barry, President and COO; John Ball, Executive VP, Operations & Administration; John Harris, Executive VP and CFO

**Hungry Minds Technology Publishing Group:** Richard Swadley, Senior Vice President and Publisher; Mary Bednarek, Vice President and Publisher, Networking; Walter R. Bruce III, Vice President and Publisher; Joseph Wikert, Vice President and Publisher, Web Development Group; Mary C. Corder, Editorial Director, Dummies Technology; Andy Cummings, Publishing Director, Dummies Technology; Barry Pruett, Publishing Director, Visual/Graphic Design

**Hungry Minds Manufacturing:** Ivor Parker, Vice President, Manufacturing

**Hungry Minds Marketing:** John Helmus, Assistant Vice President, Director of Marketing

**Hungry Minds Production for Branded Press:** Debbie Stailey, Production Director

**Hungry Minds Sales:** Michael Violano, Vice President, International Sales and Sub Rights

---

---

## ABOUT THE AUTHORS

**Danny Ryan:** Danny Ryan graduated from Georgia Tech with a degree in Industrial and Systems Engineering. Danny has more than three years of training experience and more than five years of consulting experience, most involving Fortune 500 companies. Danny has two years of “big-five” consulting experience with PricewaterhouseCoopers. Danny’s area of focus is consulting for Microsoft Internet technologies including Microsoft ASP.NET, C#, SQL Server 2000, BizTalk Server 2000, and Commerce Server 2000. Danny has used several processes for software development including variants of the Rational Unified Process and the Microsoft Solutions Framework. On projects, Danny normally plays the role of Project Manager. His certifications include MCSE, MCSA, MCT, and MCP + Internet. Danny is a recent instructor of XML, MSF Design, Microsoft Site Server 3.0, Interdev 6.0, and other Web Development classes. You can contact Danny at [dryan@threewill.com](mailto:dryan@threewill.com) or learn more about him at <http://www.threewill.com/people/dryan>.

**Tommy Ryan:** Tommy graduated from Clemson University with a degree in Chemical Engineering. Tommy has more than twelve years of technical project experience and more than four years of pure software consulting experience. Tommy’s area of focus is consulting for Microsoft Internet technologies, including Microsoft ASP.NET, C#, SQL Server 2000, BizTalk Server 2000, and Commerce Server 2000. Like Danny, Tommy has used several processes for software development, including variants of the Rational Unified Process, and the Microsoft Solutions Framework. On projects, Tommy normally plays the role of Technical Lead. His certifications include MCSE, MCSA, MCT, and MCP + Internet. Tommy is a recent instructor of MSF Design, Microsoft Site Server 3.0, Interdev 6.0, and several of the Visual Basic 6.0 classes. Tommy is also the co-author of another Visual blueprint book on building Internet application with C#. You can contact Tommy at [tryan@threewill.com](mailto:tryan@threewill.com) or learn more about him at <http://www.threewill.com/people/tryan>.

## AUTHORS’ ACKNOWLEDGMENTS

Both Danny and Tommy would like to thank Extreme Logic and all of the exceptional people that we worked with there. Special thanks to Wain Kellum, Keith Landers, Saima Adney, Alicia Ford, Bruce Harple, and Nancy Wehunt — along with all of the team members that we worked with on projects including: Anthony Yott, Brian Erwin, Dave Cohen, Alan Fraser, Jerry Rasmussen, Tim Coalson, Jim Davis, Stacy Parrish, Chris Cumpton, John Underwood, Desiree Tuvel, Rich Sabo, Teo Lachev, John Camp, Scott Matthews, Jennifer Aase, Amy Bauland, Russell Groover, Todd Ransom, David Steyer, Tony Heffner, Sean Gerety, Jason Etheridge, Julie Kellum, Ashley Aiken, and Tammy Conaway.

We would like to thank the clients that we have worked with during the past couple of years for challenging us to be better consultants, including Nick Callivas, Brian Blinco, Jay Dalke, Bob Hughes, and Harwell Thrasher.

We would also like to thank Megan Mathews and Joe Chancey of Drew Eckl & Farnham LLP, and Eddie Scott and Brad Pearce of Jackson, Reece, and Scott for the great service they provide.

We would like to thank all of the hardworking people at Hungry Minds for helping us produce this book — especially Jennifer Dorsey, Jade Williams, and our Editors. Jennifer made sure that we were taken care of well. Jade did an awesome job of editing our work — we appreciate how hard she worked on this book, and we are very thankful that she was our Production Editor.

Tommy would like to thank his colleagues at W.L. Gore and Associates who helped him start his professional career, including John Reaney, Mark Fundakowski, Diccon Bancroft, John Pyszczynski, Pamela Perdue, Erik Nightwine, Debra Raup, Ray Edmanson, Bob McCleary, Lawrence Anderson, Wolfgang Holma, and Line 10 Production Team; the WinCC Team at Siemens that helped him in his transition to being a Microsoft geek, including Emilio Matt, Rob Bohm, Bob Meads, Rich Miceli, Charlie Moore, Jörg Allmendinger, and Rene Wolf; and his extended family and friends for the support in the things outside of work including Joe and Rosemarie Markiewicz, Robert and Donna Philips, Joe and Jan Markiewicz, Chuck and Mary Hanson, Rob and Gretchen Pfeiffer, and Reverend Joe Ciccone CSP.

Danny would like to thank: his friends for their support while writing the book, especially Dan Bassett and family, Nitin Dixit, Ellen Wu, Amy Bauland, Lisa and Stephen Cox and family, Jennifer and Wyatt Gordon, Danielle and Otan Ayan, Jack and Lisa Swift and family, Chikako and Takashi Asai and family, Robin Moon, Catherine Williams, Asad Jafari, Dan and Kelly Clark and family, Darnel Barnes and family, the Harding family, the Heap family, Kitty Pinto, Wendy Marinaccio, and Erica Pearlberg; everyone involved in the Jamaica trip for their good work including Father Kevin Hargaden, Shannon Smith, Michelle Basket, Ana Nerio, and everyone who sponsored the trip; the members of the MHS 10 Year Reunion Committee for picking up my slack including Sydney Whitmer, Tina Shamblin, and Jennifer Gordon; and finally, past colleagues who have made a difference in my life including Ivan Lee, Neil Russo, Jeff Shaw, Bobby Lee, Matthew Thayer, and Steve Johnston.

---

---

We dedicate this book to our family.

Dad, thank you for teaching us by example what it means to live a life based on principles.

Mom, thank you for showing us in a real way how to live according to the most important principle, unconditional love.

Linda, thank you for having faith in us and for being the support we needed to write this book — we could not have done this without you.

Alex, Austin, Madeline, thanks for keeping us company while writing the book and keeping Kiki out of the office while we were working.

Deanna, we wish you all the best with your writing — you're the true writer in the family and we look forward to reading your book.

Bobby and Ashley, we wish you a wonderful lifetime together full of laughter, love, and great memories.

---

# TABLE OF CONTENTS

## HOW TO USE THIS BOOK .....xiv

### 1) GETTING STARTED WITH ASP.NET

Introduction to ASP.NET .....	2
Install Internet Information Server 5.0 .....	4
Install the .NET Framework .....	6
Change the Home Directory in IIS .....	8
Add a Virtual Directory in IIS .....	10
Set a Default Document for IIS .....	14
Change Log File Properties for IIS .....	16
Stop, Start, or Pause a Web Site .....	18

### 2) WEB DEVELOPMENT BASICS

Browse Your Default Web Site .....	20
Explore a Web Site .....	22
Open a Template File .....	24
Save a File to the Default Web Site .....	25
Create an HTML Page .....	26
Create an ASP Page .....	28
Create an ASP.NET Web Page .....	30
Add an Event Handler to an ASP.NET Page .....	32

### 3) C# BASICS

Write Your First C# Application .....	34
Compile a C# Application .....	36
Format Your Code .....	38
Declare a Variable .....	40
Initialize a Variable .....	42
Access Properties .....	44
Make Decisions Using Conditional Statements .....	46
Work with Arrays .....	48
Control Logic Using Iterative Statements .....	50
Concatenate a String .....	52
Convert a Variable .....	54
Enumerate a Collection .....	56

Declare and Use Methods .....	58
Implement Exception Handling .....	62
Convert a Console Application to an ASP.NET Web Page .....	64

## 4) WORK WITH HTML CONTROLS

---

Introduction to HTML Controls .....	66
Process Requests to the Server .....	68
Create a Form Button .....	70
Create an HTML 4.0 Button .....	72
Create a Graphical Button .....	73
Request Single Line Input .....	74
Request Multiple Line Input .....	76
Request Boolean Input .....	77
Request a Selection from a Group .....	78
Request Input from a Drop-Down List .....	79
Create a Link .....	80
Render an Image .....	82
Build a Table .....	84
Store Hidden Information on a Form .....	86
Upload Files .....	88

## 5) WORK WITH WEB CONTROLS

---

Introduction to Web Controls .....	90
Create a Button for Posting Data .....	92
Create a Hyperlinked Button .....	94
Create a Graphical Button .....	95
Request Text Input .....	96
Request Boolean Input .....	98
Request a Selection from a Group .....	100
Request Input from a Drop-Down List .....	102
Request Dates from a Calendar .....	104
Create a Link .....	106



## 5) WORK WITH WEB CONTROLS (CONTINUED)

Render an Image .....	107
Build a Table .....	108
Manipulate Text .....	110
Add a Placeholder for Controls .....	111
Provide a Container for Controls .....	112
Display Advertisement Banners .....	114
Validate Required Fields .....	116
Compare Two Fields for Validation .....	118
Check the Boundaries of Input .....	120
Validate with Regular Expressions .....	122
Summarize Validation Errors .....	124

## 6) ACCESS DATA WITH ASP.NET

Introduction to Data Access with ASP.NET .....	126
Display Repeating Data .....	128
Display Complex Lists .....	130
Display SQL Data .....	132
Insert Data into a SQL Database .....	134
Update Data from a SQL Database .....	136
Delete Data from a SQL Database .....	138
Sort Data from a SQL Database .....	140
Execute Stored Procedures .....	142
Work with Master-Detail Relationships .....	144
Work with XML Data Sources .....	146
Transform and Display XML .....	148

## 7) WORK WITH WEB SERVICES

Introduction to Web Services.....	150
Create a Simple Web Service .....	152
Test a Web Service .....	154
Using a Parameter with a Web Service .....	156
Return an Array from a Web Service .....	158
Return an Enumerated Type from a Web Service .....	160
Return an Object from a Web Service .....	162
Return XML from a Web Service .....	164

Return SQL Data from a Web Service .....	166
Work with the Session Object in a Web Service .....	168
Work with the Application Object in a Web Service .....	170
Create a Client Web Page for a Web Service .....	172
Create a Client Console Application for a Web Service .....	174

## 8) CREATE CUSTOM COMPONENTS

---

Create a Simple Component .....	176
Create a Stateful Component .....	180
Create a Two-Tier Web Form .....	184
Create a Three-Tier Web Form .....	188
Use a Code-behind for Your ASP.NET Page .....	192

## 9) USING ASP.NET COMPONENTS

---

Read Form Data with Request.Form .....	194
Display Data with Request.Params .....	196
Write Output Using Response.Write .....	198
Redirect Using Response.Redirect .....	200
Check for Web Browser Types .....	202
Send an E-Mail Using ASP.NET .....	204
Use the ASP.NET Page Cache .....	206
Use the ASP.NET Data Cache .....	208

## 10) ASP.NET APPLICATIONS AND STATE MANAGEMENT

---

Introduction to Applications and State Management .....	210
Create a Global.asax File .....	212
Using Processing Directives in the Global.asax file .....	216
Using Server-Side Objects in the Global.asax File .....	218
Using Application Event Handlers in the Global.asax File .....	220
Using Application State .....	222
Using Session State .....	226
Work with Cookies .....	230
Work with Page State .....	234

# TABLE OF CONTENTS

## 11) CONFIGURE YOUR ASP.NET APPLICATIONS

Add Application Settings .....	236
Set Standard Configuration .....	238
Add Custom Settings .....	240

## 12) DEBUG YOUR ASP.NET APPLICATIONS

Enable Page-Level Debugging .....	242
Enable Custom Error Handling .....	244
Handle Errors Programmatically .....	246
Use a Page-Level Trace .....	248
Use an Application-Level Trace .....	250

## 13) SECURITY AND ASP.NET

Using Windows Authentication .....	252
Using Forms Authentication .....	256
Authorize Users .....	260

## 14) LOCALIZATION AND ASP.NET

Set Up Encoding .....	262
Using CultureInfo .....	264
Using RegionInfo .....	266
Localize with the Page Control .....	268
Create and Use Resources .....	270
Use Resource Manager Information .....	272

## 15) MIGRATE FROM ASP TO ASP.NET

Work with Multiple Server-Side Languages .....	274
Work with Script Blocks .....	276
Using Render Functions .....	278
Using Page Directives .....	280
Migrate VBScript to VB.NET Syntax .....	282
Migrate JScript to JScript.NET Syntax .....	286

<b>APPENDIX A: ASP.NET QUICK REFERENCE</b> .....	288
<b>APPENDIX B: C#, VB, AND JSCRIPT LANGUAGE EQUIVALENTS</b> .....	292
<b>APPENDIX C: WHAT'S ON THE CD-ROM</b> .....	300
<b>HUNGRY MINDS END-USER LICENSE AGREEMENT</b> .....	304
<b>INDEX</b> .....	306

# HOW TO USE THIS BOOK

*ASP.NET: Your visual blueprint for creating Web applications on the .NET framework* uses straightforward examples to teach you many of the tasks required to write Web applications on the .NET framework.

To get the most out of this book, you should read each chapter in order, from beginning to end. Each chapter introduces new ideas and builds on the knowledge learned in previous chapters. Once you become familiar with ASP.NET, this book can be used as an informative desktop reference.

## Who This Book Is For

If you are looking for a resource that will help you quickly get started creating ASP.NET Web pages, *ASP.NET: Your visual blueprint for creating Web applications on the .NET framework*. This book will walk you through the basics that you need to get started and familiarize yourself with the essentials of working with ASP.NET. This book also demonstrates advanced features of ASP.NET, such as creating custom components, configuration, debugging, security, and migration to ASP.NET from ASP 3.0.

No prior experience with ASP.NET is required, but familiarity with the operating system installed on your computer is an asset.

Experience with programming languages is also an asset, but even if you have no programming experience, you can use this book to learn the essentials you need to work with ASP.NET. The C# programming language is used for most of the tasks in this book.

## What You Need To Use This Book

To perform the tasks in this book, you need a computer with an operating system on which you can install the .NET framework. The tasks in this book are developed using Windows 2000 with IIS 5.0 installed. The computer will also need to have a text editor to create code, such as Notepad. Visual Studio 7.0 is not required to do the tasks in this book, although you can use it if you want. Many of the tasks in this book require a Web browser, such as Internet Explorer.

## The Conventions In This Book

A number of typographic and layout styles are used throughout *ASP.NET: Your visual blueprint for creating Web applications on the .NET framework* to distinguish different types of information.

Courier Font

Indicates the use of C#, VB, or Jscript variable names, keywords, and other elements of ASP.NET code.

**Bold**

Indicates information that you must type.

*Italics*

Indicates a new term being introduced.

**Apply It**

An Apply It section usually contains a segment of code that takes the lesson you just learned one step further. Apply It sections offer inside information and pointers that can be used to enhance the functionality of your code.

**Extra**

An Extra section provides additional information about the task you just accomplished. Extra sections often contain interesting tips and useful tricks to make working with ASP.NET easier and more efficient.

## The Organization Of This Book

*ASP.NET: Your visual blueprint for creating Web applications on the .NET framework* contains 15 chapters and three appendices.

The first chapter, Getting Started with ASP.NET, explains how you can install the .NET framework and configure many of the options available when setting up your Web site.



Chapter 2, *Web Development Basics*, presents the fundamentals of working with ASP.NET including exploring and browsing your Web site, opening and saving files, and creating your first ASP.NET Web page. This chapter also shows you how to use the sample program templates on the CD-ROM.

Chapter 3, *C# Basics*, introduces you to the C# programming language, including how to work with variables, conditional statements, arrays, looping structures, strings, collections, and exception handling. This will prepare you for the material in later chapters if you are not familiar with the C# programming language.

Chapter 4, *Work with HTML Controls*, gets you started with creating Web pages in ASP.NET that have buttons, text boxes, check-boxes, drop-down lists, and tables.

Chapter 5, *Work with Web Controls*, will show you how to add many of the common controls to Web pages like text boxes using ASP.NET Web controls. The chapter then demonstrates how to add special controls like calendars and advertisement banners. Finally, the chapter explains how to validate user input in ASP.NET.

Chapter 6, *Access Data with ASP.NET*, explains how to use some of the controls you will use to display data like the Repeater and the DataGrid control. You will see how to insert, update, delete, and sort data. The chapter also covers working with stored procedures and XML.

Chapter 7, *Work with Web Services*, explains how you can create, test, and consume Web services. The chapter explains how to return a number of data types including arrays, enumerations, objects, XML, and SQL Data. You see how to create clients for the Web service, including a Web page client and a Console client.

Chapter 8, *Create Custom Components*, explains how to create components, build a two-tier and a three-tier Web Form, and store code in Code-behind components for your ASP.NET Web pages.

Chapter 9, *Using ASP.NET Components*, explains how to work with components not covered in other chapters including the Browser Capabilities component and components necessary to send e-mail. Page and Data caching of your ASP.NET Web pages is covered as well.

Chapter 10, *ASP.NET Applications and State Management*, explains how to work with application and session state with ASP.NET. It also covers working with cookies and page state.

Chapter 11, *Configure Your ASP.NET Applications*, describes the process for setting and retrieving configuration information for your ASP.NET application.

Chapter 12 introduces you to debugging your ASP.NET applications.

Chapter 13, *Security and ASP.NET*, shows you several different ways to secure your ASP.NET applications.

Chapter 14, *Localization and ASP.NET*, walks you through how to globalize your ASP.NET application for different cultures, locales, and languages.

The final chapter explains some of the details about migrating to ASP.NET, including migrating code from VBScript to VB.NET and other important issues to address when migrating.

The Appendices include useful tables of reference material and a summary of important elements of ASP.NET syntax and the C#, VB, and Jscript languages.

## What Is On The CD-ROM

The CD-ROM in the back of this book contains the sample code from each of the two-page lessons, as well as the code from most of the Apply It sections. This saves you from having to type the code and helps you quickly get started creating ASP.NET programs. The CD-ROM also contains several shareware and evaluation versions of programs that can help you work with ASP.NET. An e-version of this book is also available on the companion disc.

# INTRODUCTION TO ASP.NET

**A**SP.NET is a programming framework developed by Microsoft for building powerful Web applications.

## Web Servers

The previous version of Active Server Pages was ASP 3.0. ASP.NET and ASP 3.0 can both run on Internet Information Server (IIS) 5.0 with Windows 2000 and Windows XP. You can have your ASP.NET and your ASP 3.0 applications run on the same server.

If you use Windows 95, 98, ME, or NT, and you want to run ASP.NET applications, you can install Windows NT or XP in addition to your other operating system by creating a dual-boot machine. This will enable you to run two operating systems on one machine, giving you the ability to run ASP.NET and keeping your original operating system intact. You will have to devote around 5GB of disk space to install the operating system (OS), the .NET Framework SDK, and any other supporting applications, such as SQL Server 2000. To separate the files associated for each OS, you should create a separate partition for the new OS.

## Versions

The ASP.NET Framework is supported on Windows 2000 and Windows XP. ASP.NET applications will run on IIS 5.0 for these operating systems.

Web Services is supported on all platforms supported by the Microsoft .NET Framework SDK, except Windows 95.

Windows XP, Windows 2000, Windows NT 4 with Service Pack 6a, Windows ME, Windows 98, Windows 98 SE, and Windows 95 all support the Microsoft .NET Framework SDK.

## Language Support

ASP.NET has built-in support for three languages: Visual Basic (VB), C#, and JScript. You can install support for other .NET-compatible languages as well.

## Tools

Microsoft designed ASP.NET to work with WYSIWYG HTML editors and other programming tools. Or, you can even use a simple text editor like Notepad. The Notepad text editor is used in this book's code samples. If you want more support from your development environment for coding, you can use Microsoft Visual Studio.NET. Using a tool such as Microsoft Visual Studio.NET enables you to take advantage of other features such as GUI support of drag and drop Server Controls and debugging support.

## Web Forms

ASP.NET Web Forms gives you the ability to create Web pages on the .NET platform. Web Forms enable you to program against the controls that you put on your Web pages. You can either use a Server Control that is built into ASP.NET or create your own custom Server Controls. These Server Controls are used for controlling HTML tags on a Web page. By using Web Forms, you can build user interface code as effectively as your Business Services code, reusing and packaging the code in a well-designed manner.

## Web Services

ASP.NET Web Services gives you the ability to access server functionality remotely. Using Web Services, businesses can expose their data and/or component libraries, which in turn can be obtained and manipulated by client and server applications. Web Services enable the exchange of data in client-server or server-server scenarios, using standards like HTTP and XML messaging to move data across firewalls. Web services are not tied to a particular component technology or object-calling convention. As a result, programs written in any language, using any component model, and running on any operating system can access Web services.

### State and Application

ASP.NET provides a simple framework that enables Web developers to write logic that runs at the application level. Developers can write this code in either the `global.asax` text file or in a compiled class. This logic can include application-level events, but developers can easily extend this framework to suit the needs of their Web application. ASP application code, written in the `global.asa` file, is supported in ASP.NET. You can simply rename `global.asa` to `global.asax` when upgrading from ASP.

### Data Access

Accessing databases from ASP.NET applications is a common technique for displaying dynamic information to Web site visitors. ASP.NET makes it easier than ever to access databases for this purpose and provides for managing the data in the database.

### Performance

A big difference between ASP 3.0 and ASP.NET is how your code is run on the server. With ASP.NET, your code is compiled into executable classes. With ASP 3.0, code often needs to be interpreted. With ASP 3.0, any server-side code is most likely going to have to be interpreted by the Web server, unless it is cached. If you want to avoid interpreted code in ASP 3.0, you need to put the code into a COM component.

### Power

With ASP.NET, you now have access to the *common language runtime* (or *CLR*). Running on the CLR gives access to many of the features available in the .NET Framework, such as debugging, security, data access, language interoperability, and more.

### Configuration

ASP.NET configuration settings are stored in XML-based files, which are text files easily accessible for reading and writing. Each of your applications can have a distinct configuration file. You can extend the configuration scheme to suit your requirements.

### Security

The .NET Framework and ASP.NET provide default authorization and authentication schemes for Web applications. You can easily remove, add to, or replace these schemes depending upon the needs of your application.

### Migration from ASP to ASP.NET

Simple ASP pages can easily be migrated to ASP.NET applications. ASP.NET offers complete syntax and processing compatibility with ASP applications. Developers simply need to change file extensions from `.asp` to `.aspx` to migrate their files to the ASP.NET framework. They can also easily add ASP.NET functionality to their applications with ease, sometimes by simply adding just a few lines of code to their ASP files. For additional information on handling migration issues, see page 282.

# INSTALL INTERNET INFORMATION SERVER 5.0

**M**icrosoft Internet Information Server (IIS) is the Web server software that you can use to create, administer, and host Web sites for the ASP.NET framework. You can install the software from all versions of the Windows 2000 CD-ROM disc. Because Internet Information Server 5.0 is installed by default when installing Windows 2000 in a new installation, you may not need to install the software. If you have upgraded to Windows 2000 from a previous version of Windows, make sure that Internet Information Server 5.0 exists after the upgrade process.

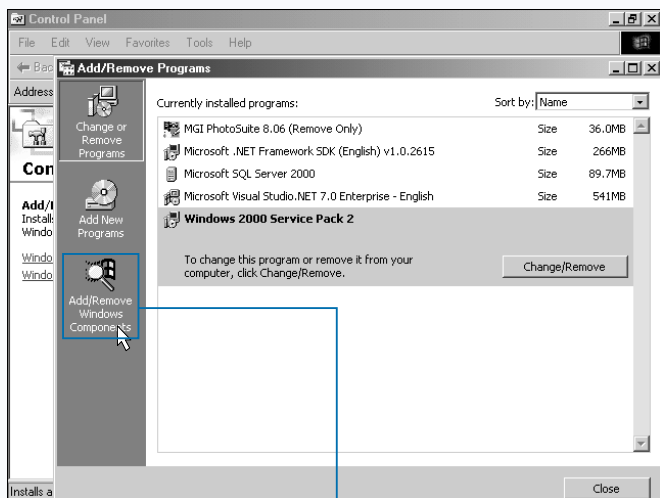
IIS comes as a Windows Component and can be installed from the Windows Component Wizard.

The Windows Component Wizard becomes available when you select Add/Remove Programs from the Control Panel.

In this wizard, you should install at least the Common Files, which installs the programs you need to run IIS 5.0: IIS Snap-In, which is an interface for administering your Web sites, and World Wide Web Server, which enables users to access your Web sites.

After installing IIS 5.0, restart the Web server. IIS 5.0 services are set to automatically start when your computer reboots, so the software will be ready for you to install the .NET Framework upon rebooting.

## INSTALL INTERNET INFORMATION SERVER 5.0



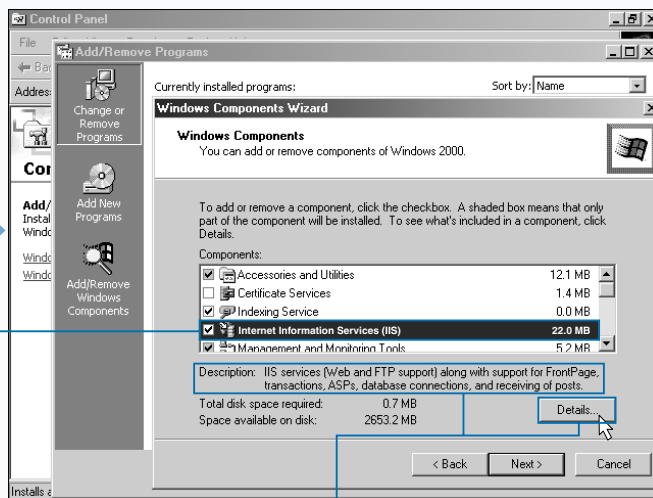
**1** Insert the Windows 2000 CD-ROM into your CD-ROM drive.

*Note: If the Microsoft Windows 2000 dialog box appears, close the dialog box.*

**2** In the Control Panel, double-click Add/Remove Programs.

The Add/Remove Programs window appears.

**3** Click Add/Remove Windows Components.



The Windows Components Wizard dialog box appears listing the installation size of the component you can add or remove.

**4** Click  next to Internet Information Services to select it ( changes to .

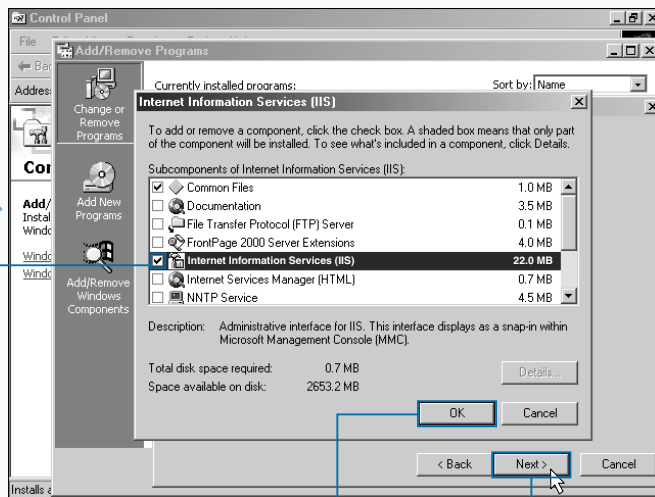
This area displays a description of the Internet Information Services subcomponents.

**5** Click Details.

**Extra**

You have several other IIS subcomponents that you can install. If you choose Documentation, it will install help files and samples to assist you with administering IIS. File Transfer Protocol (FTP) Server installs support for uploading and downloading files using FTP sites. If you are using Visual Studio or FrontPage for working with any of your sites, you may want to install FrontPage 2000 Server Extensions. The Internet Services Manager (HTML) is a Web-based version of the administration functionality that enables you to administer the server using a browser. Install the NNTP service if you need support for network news. Install the SMTP service if you need support for e-mail functionality.

You can administer IIS through the Internet Services Manager, which is available from Start ⇨ Programs ⇨ Administrative Tools ⇨ Internet Services Manager. Because this tool is designed as a Microsoft Management Console Snap-In, you can add it to your own custom console along with any other Snap-Ins that you use often.

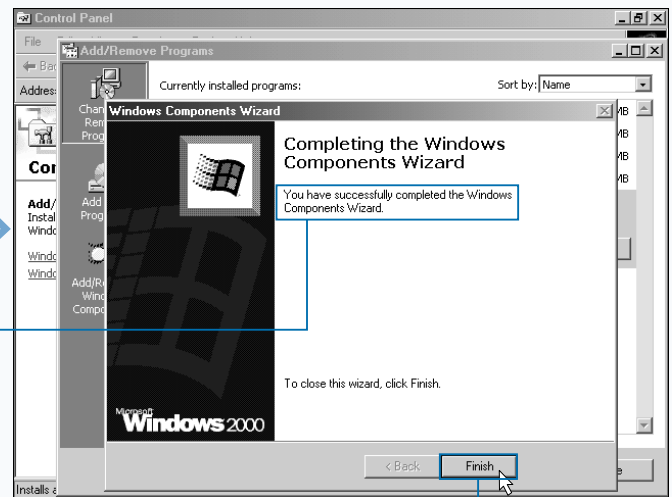


■ The Internet Information Services (IIS) dialog box opens.

6 Click  next to each subcomponent you want to install ( changes to .

7 Click OK to confirm your selections and close the dialog box.

8 Click Next to install the parts of the Internet Information Services and subcomponents you selected.



■ When the installation is complete, a message appears confirming the successful installation of IIS.

9 Click Finish to close the Wizard.

*Note: You should now restart your computer.*



# INSTALL THE .NET FRAMEWORK

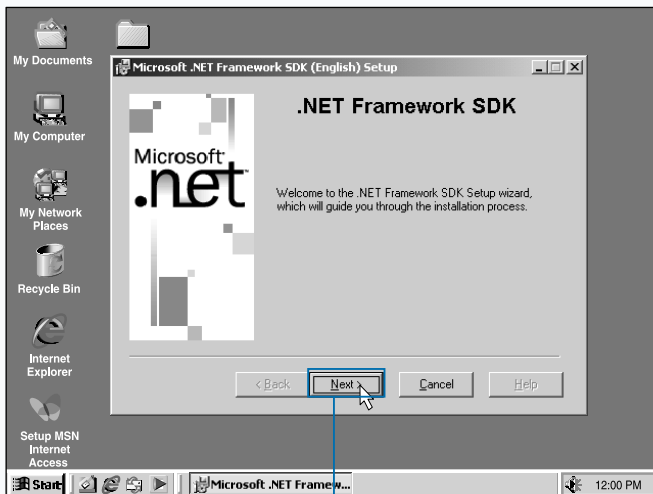
After you have the IIS installed on your Web server, you can install the Microsoft .NET Framework SDK. You can take two paths to get the .NET Framework installed on your machine. The first path is to install the .NET Framework SDK. This install includes the necessary runtime to process your ASP.NET applications and also has documentation for using the .NET Framework. The second way to get the Microsoft .NET Framework installed on your machine is to install Visual Studio.NET. This path gets the .NET Framework along with Microsoft's development tool for ASP.NET applications.

The process for installing the .NET Framework SDK starts with obtaining the setup program. When you

run the setup program, you may be required to update certain software on your server. For instance, your version of the Microsoft Data Access Components (MDAC) may need updating to a more recent version. The installation process includes accepting a software agreement, specifying which parts of the .NET SDK to install, and specifying where the .NET SDK is installed.

When installing .NET SDK make sure that you have enough hard drive space for the installation. As of Beta 2, you need 311MB to complete the installation.

## INSTALL THE .NET FRAMEWORK

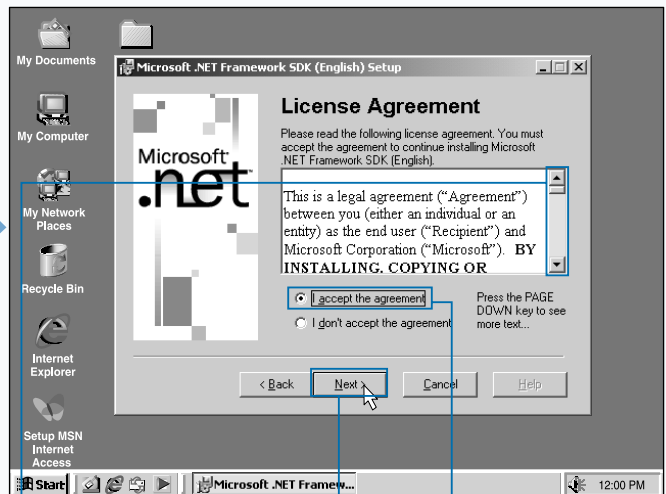


**1** Insert the CD-ROM with the .NET SDK and run the setup.exe file.

*Note: This task assumes that you have the installation program on CD-ROM. You can also install the .NET SDK from a local drive or a network drive.*

**2** The .NET Framework SDK Setup page appears.

**2** Click Next to continue.



**3** The license agreement for the software appears in the middle of the dialog box. You may need to scroll down to see the entire license agreement.

**3** Click  next to I accept the agreement to accept the software license ( changes to .

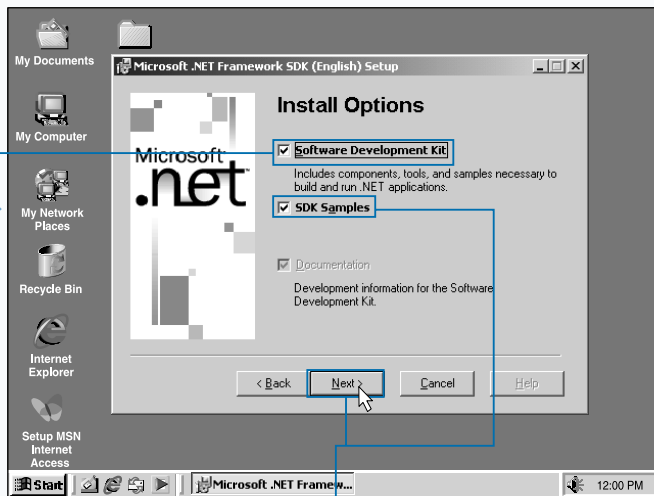
**4** Click Next to continue.

**Extra**

You can download the latest .NET Platform SDK from <http://msdn.microsoft.com/downloads/>. This task uses Build 1.0.2914 of the .NET Platform SDK. You can also go to the Microsoft newsgroups for the .NET Platform SDK by going to <http://msdn.microsoft.com/newsgroups/>.

Before installing the .NET Platform SDK, be sure to check the release notes for any special instructions. These notes will let you know what issues are present in the build that you are installing. The availability of namespaces on each operating system changes quite often. For example, going from Beta 1 to Beta 2 of the .NET Platform SDK, ASP.NET and transactions for managed code support were dropped for Windows NT 4.0.

Be careful when searching the Web for source code that runs on the .NET Runtime. The namespace names have changed often, especially from Beta 1 to Beta 2. In particular, data access framework classes changed dramatically.

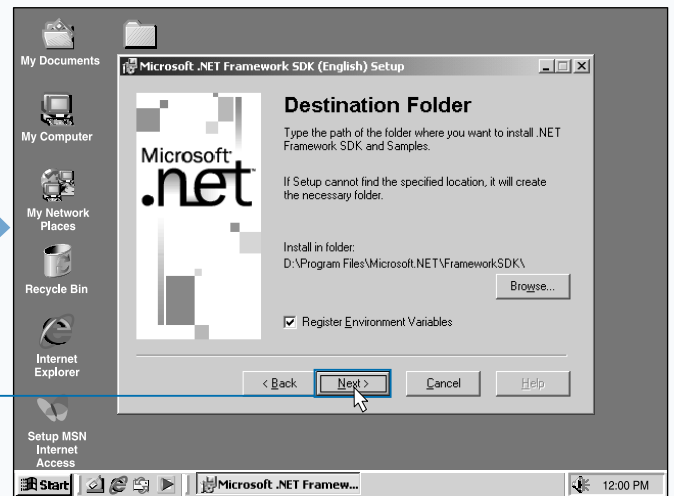


■ A dialog box appears allowing you to specify whether to install the .NET software, samples, and documentation.

5 Click  next to Software Development Kit to select it ( changes to .

6 Click  next to SDK Samples to select it ( changes to .

7 Click Next to continue.



■ The Destination Folder window appears.

8 Click Next to select the default folder and to continue.

■ The .NET SDK is installed.

# CHANGE THE HOME DIRECTORY IN IIS

The location where the files for your Web site are stored is called the *home*, or *root*, directory. You can specify which directory on your Web server is the home directory. Any files that are in any of the subdirectories of the home directory will be available as well. Make sure that this home directory, along with any of its subdirectories, has proper file system security applied. If someone bypasses the security of IIS, you must make sure that they only have read access permissions to the files. There are only a few exceptions to this rule.

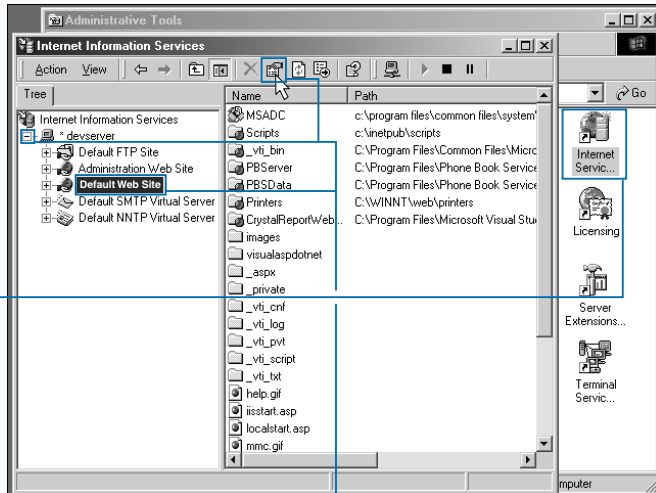
You can also set up virtual directories to make other directories that are not underneath your home

directory available for your Web site. See page 10 for more information on virtual directories.

The default location for the Web site that is created when installing IIS is `C:\inetpub\wwwroot\`. You can change this location to be another directory of your choice.

When users come to your Web site, the document that they first see is the default document. The default document must be placed in the home directory. See page 14 for information on setting the default document.

## CHANGE THE HOME DIRECTORY IN IIS



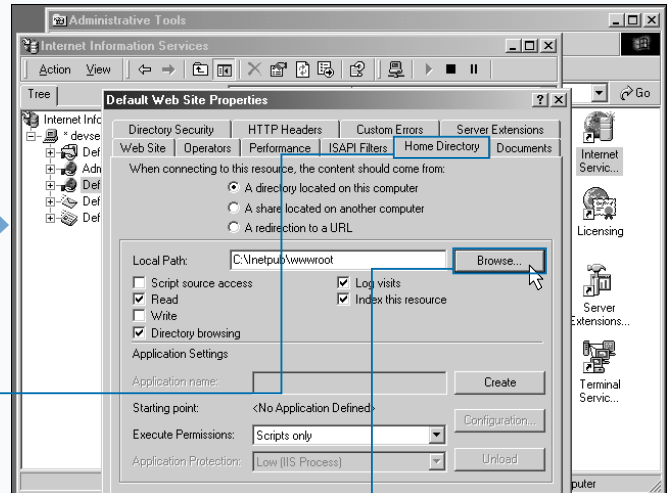
**1** In the Control Panel, double-click Administrative Tools to open the Administrative Tools window.

**2** Double-click Internet Services Manager to open the Internet Information Services window.

**3** Click  $\oplus$  to expand the list of Web sites on the Web server ( $\oplus$  changes to  $\ominus$ ).

**4** Click the Web site whose home directory you want to change.

**5** Click the Properties button  $\left(\text{Ⓜ}\right)$ .



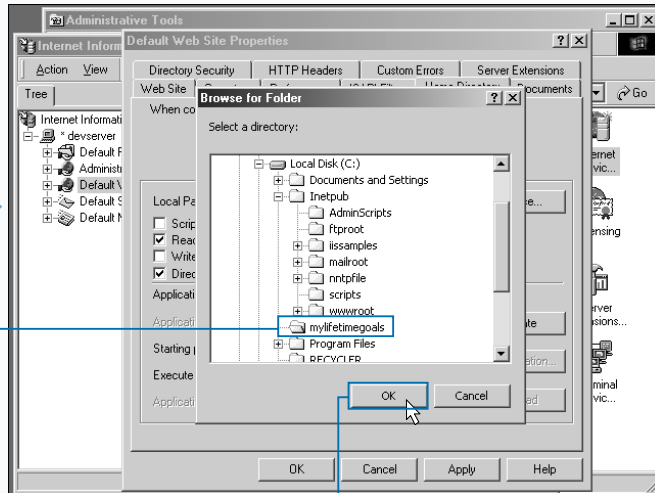
**6** The Default Web Site Properties dialog box appears.

**7** Click Browse to open the Browse for Folder dialog box.

**Extra**

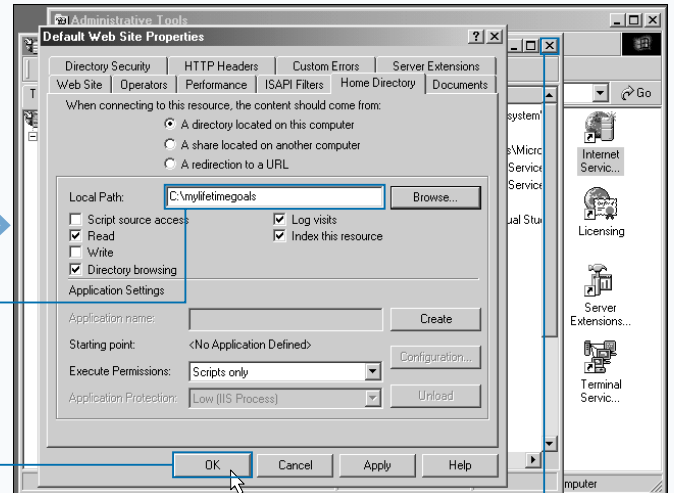
There are several options you can choose when specifying a home directory. Follow steps 1 to 5 on page 8 and then select the option you want to use. Use the table below to determine which option is best for setting the location of the home directory.

OPTION	DESCRIPTION
A directory located on this computer	Use this option for specifying a directory located on the Web server as the home directory. This is the most common choice.
A share located on another computer	Use this option if you want to specify a share directory as the home directory. You can specify the user that you want to connect as using the Connect As button.
A redirection to a URL	Use this option if you want to have users redirected to another URL when they are trying to access any part of your site.



**8** Click the directory you want to set as your new home directory.

**9** Click OK to set the new home directory and to close the dialog box.



The new home directory appears in the Local Path.

**10** Click OK to close the properties dialog box.

**11** Close the Internet Information Services window.

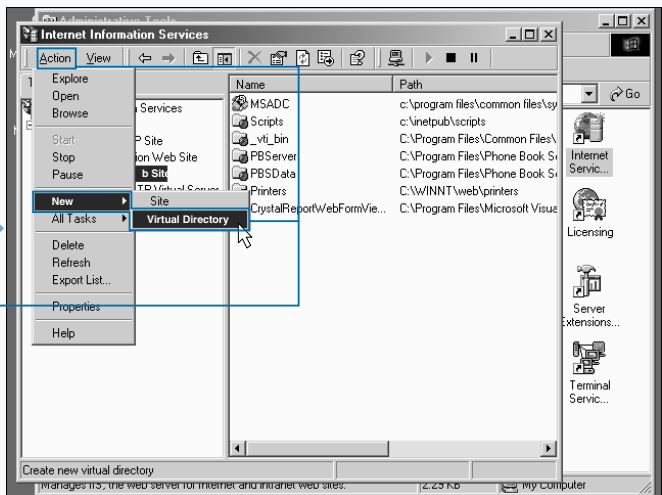
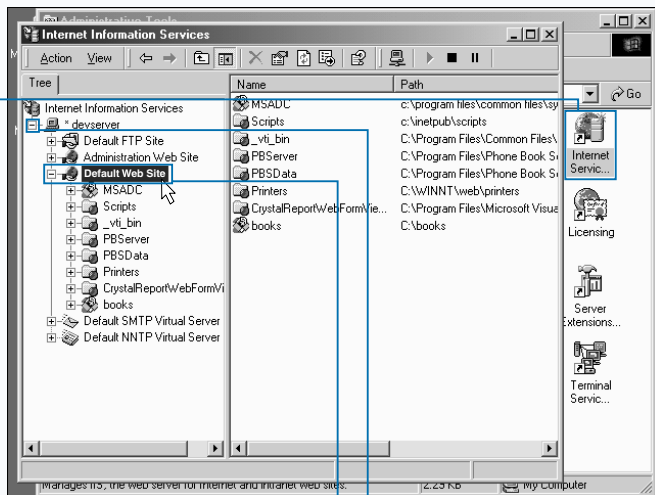
# ADD A VIRTUAL DIRECTORY IN IIS

After you have installed Internet Information Server (IIS), you can configure your Web sites. Each Web site will have its own home directory. For example, the default Web site that is created when installing IIS will have a home directory located at `C:\inetpub\wwwroot`. All of the subdirectories of this directory will be accessible to your users. For example, suppose that your site is available at `www.mylifetimegoals.com`. You add a subdirectory under your default directory (for example, `C:\inetpub\wwwroot\test`). After that subdirectory is added, you access the files in that directory using `www.mylifetimegoals.com/test` as your URL.

Virtual directories give you the ability to access directories that are not necessarily a subdirectory under your home directory. For example, you can create a virtual directory with the alias `virtual` and map it to any physical directory (for example, `C:\virtual`). You can then access the files in `virtual` by using the address `www.mylifetimegoals.com/virtual`.

Having virtual directories is helpful if you have files located on another server. You may have files on one server that are shared by sites that are distributed across multiple Web servers. This sharing can be accomplished by accessing this shared location through a virtual directory.

## ADD A VIRTUAL DIRECTORY IN IIS



**1** In the Control Panel, double-click Administrative Tools to open the Administrative Tools window.

**2** Double-click Internet Services Manager to open the Internet Information Services window.

**3** Click  $\oplus$  to expand the list of Web sites on the Web server ( $\oplus$  changes to  $\ominus$ ).

**4** Click the Web site in which you want to add the virtual directory.

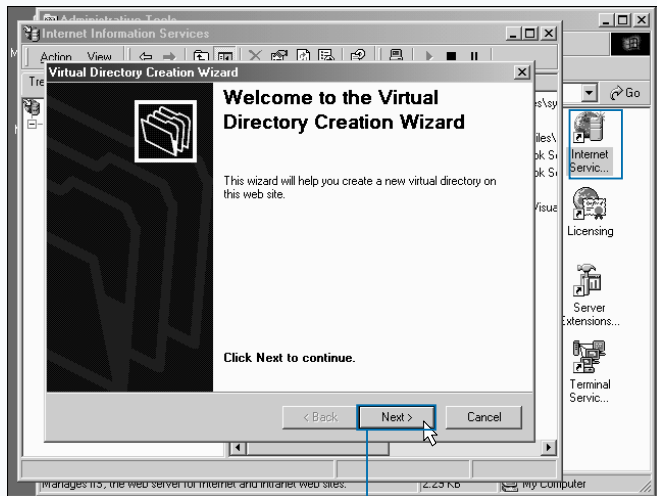
**5** Click Action  $\Rightarrow$  New  $\Rightarrow$  Virtual Directory.



**Extra**

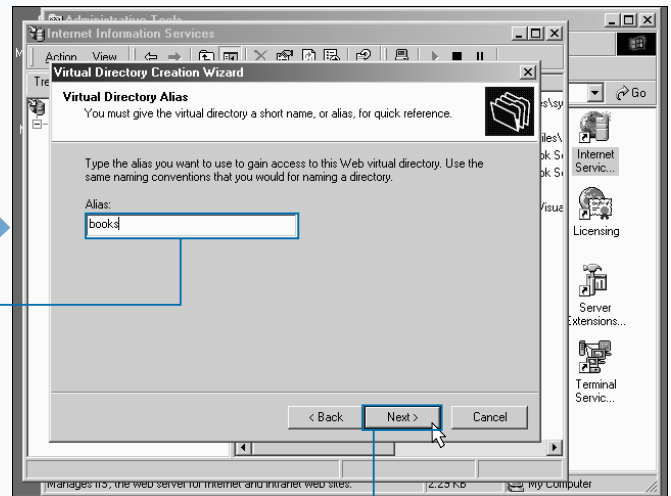
Virtual directories can also be set using Windows Explorer. Select the folder you want to set up as a virtual directory in Windows Explorer. Click the File Menu, then choose Properties. Then click the Web Sharing tab and select the Web site you want to make the virtual directory with. Click the Share This Folder option. In the Edit Alias dialog box, specify an alias and set the appropriate access and application permissions. Click OK when you finish. Then click OK again to close out the Properties dialog box.

You can also remove a virtual directory you no longer need from IIS. Perform steps 1 through 3 to open the Internet Information Services window, and then select the virtual directory you want to remove. Click the Delete button and then click Yes in the dialog box that appears. Removing a virtual directory does not remove the actual directory and its contents from the computer.



■ The Virtual Directory Creation Wizard appears.

**6** Click Next to continue.



**7** Type an alias for the virtual directory.

**8** Click Next to continue.

CONTINUED

# ADD A VIRTUAL DIRECTORY IN IIS

After you specify the path to the directory that you want to set up as a virtual directory, you can set access permissions for the directory. You have a choice of five settings to choose from or combine.

The Read permission enables users to access Web pages. Select this setting for a virtual directory containing contents that you want to make available for users to view. The Read permission is turned on by default.

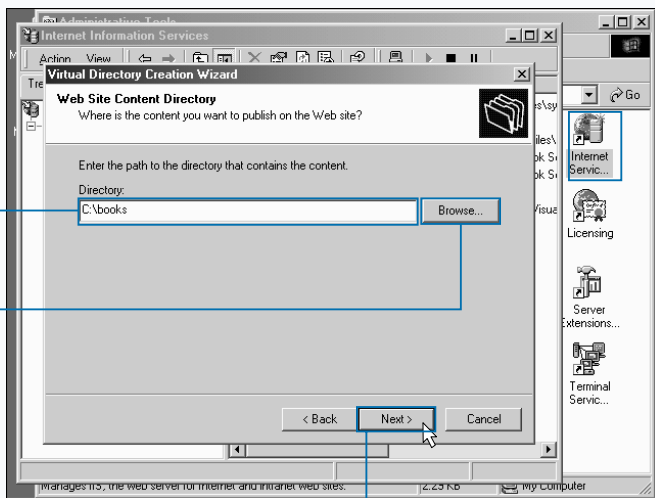
The Run scripts permission enables scripts to run in the directory. Select this setting for virtual directories containing ASP pages. The Run scripts permission is turned on by default.

The Execute permission enables applications to run in the virtual directory. For security reasons, the Execute permission is rarely enabled.

The Write permission enables the creation of files in the directory. Select this setting for virtual directories that have ASP pages that create files on the server side.

The Browse permission enables users to view the list of all the subdirectories and pages that the virtual directory contains. When a user specifies a URL that contains a directory and does not specify the name of the page, then a list of files and subdirectories appears. If there is a default document in that directory, it will appear instead of the directory listing.

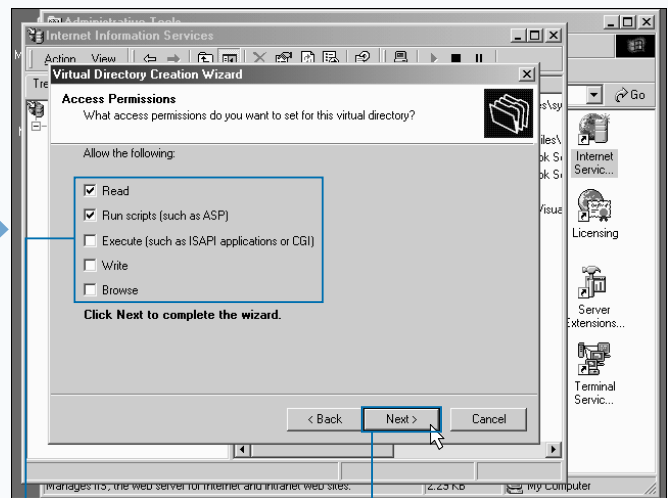
## ADD A VIRTUAL DIRECTORY IN IIS (CONTINUED)



**9** Type the path to the directory in which you want to create a virtual directory.

You can also click Browse to locate the directory on your computer.

**10** Click Next to continue.



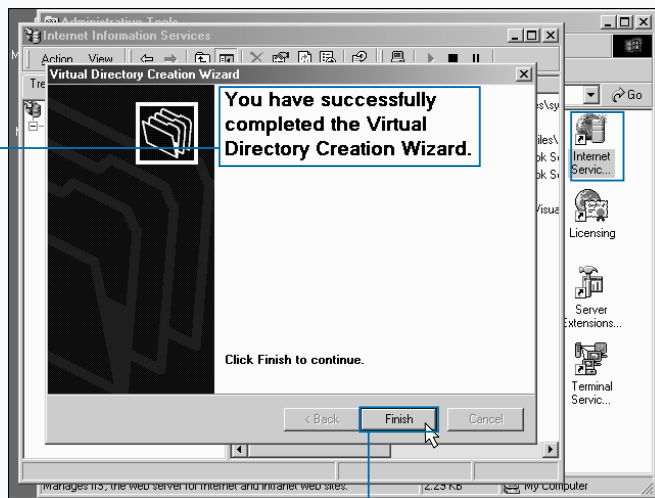
**11** Click  next to the appropriate access permissions ( changes to .

**12** Click Next to continue.

**Extra**

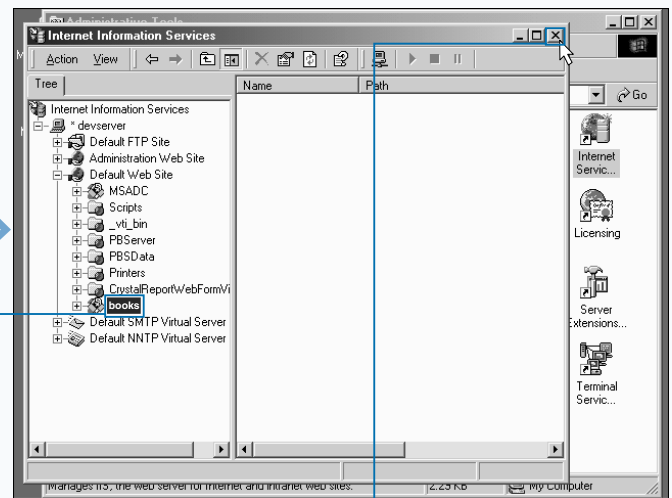
Like home directories in Web sites, virtual directories can be specified as either a directory on a Web server, a share on another computer, or as a redirection to a URL. The option to create a virtual directory map to a network share or URL is not available from the virtual directory creation Wizard. To work around this, initially set out the virtual directory to be mapped to the directory. Then, change this by accessing the properties of the virtual directory and specifying a network share or URL.

You can also use virtual directories to isolate unstable sections of your Web application. A virtual directory enables you to choose the application protection level, unlike normal subdirectories of the Web site. Choose High (Isolated) for virtual directories that contain code that needs isolation from the rest of the site.



The wizard confirms that you have successfully created a new virtual directory.

**13** Click Finish.



The new virtual directory appears under the Web site you selected.

**14** Click X to close the Internet Information Services window.

# SET A DEFAULT DOCUMENT FOR IIS

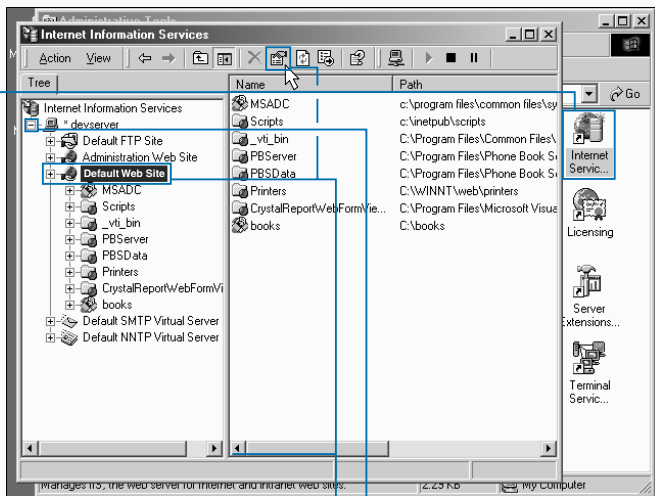
The document that is sent to a user's browser, if no specific page is requested, is called the *default document*. You can specify zero to many filenames for IIS to look for when searching for the default document. If IIS does not find the document that matches the specified filenames, it sends back an error message. When directory browsing is set for the requested directory, the user sees a list of files and subdirectories instead of an error.

You can specify a different set of default documents for each directory in a Web site. When specifying a default document, you can also set the order in which

IIS looks for a match. It starts from the top of the list and searches downward, sending the first document it finds.

You should keep in mind some of the de facto standards when choosing the list of filenames to use in the list of default documents. Filenames like `index.htm`, `default.aspx`, `default.asp`, `default.htm`, `default.html`, and `home.htm` are some of the most common filenames used for default documents. Sticking with this as a standard will help when it comes to troubleshooting the site.

## SET A DEFAULT DOCUMENT FOR IIS



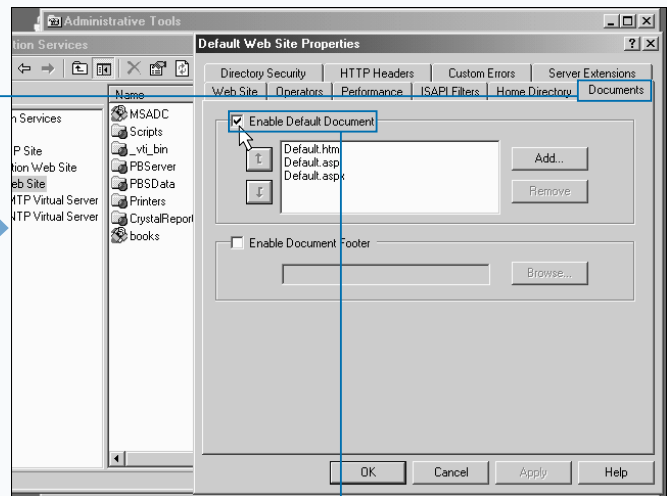
**1** In the Control Panel, double-click Administrative Tools to open the Administrative Tools window.

**2** Double-click Internet Services Manager to open the Internet Information Services window.

**3** Click **+** to expand the list of Web sites on the Web server (**+** changes to **-**).

**4** Click the Web site that you want to set as your default document.

**5** Click the Properties button (key icon).



**6** The Default Web Site Properties dialog box appears.

**7** Click the Documents tab.

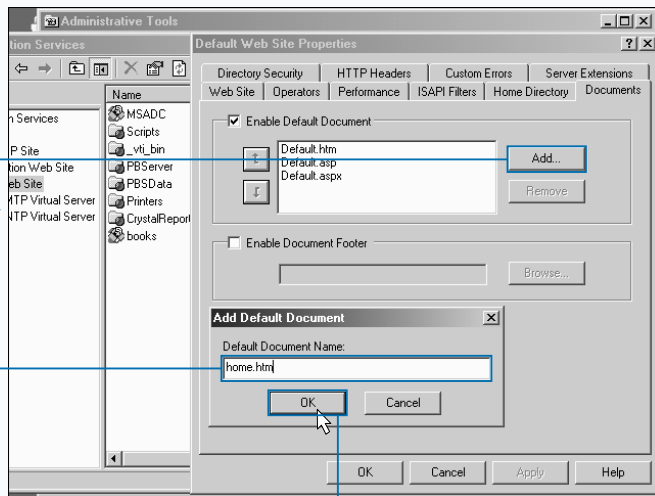
**8** Click  Enable Default Document to select it ( changes to ).

**Extra**

Another option on the Documents tab for the Web Site Properties page is the document footer. This option is an easy and convenient way to put a footer at the bottom of all documents on your site. To do this, create a file with some HTML formatting. You should not put HTML tags such as <TITLE> or <BODY>, because they are already a part of the pages in your site. Perform steps 1 to 6 below to display the Documents tab of the Default Web Site Properties dialog box. Click Enable Document Footer and then click the Browse button to locate the HTML file you created.

**HTML Footer File Example:**

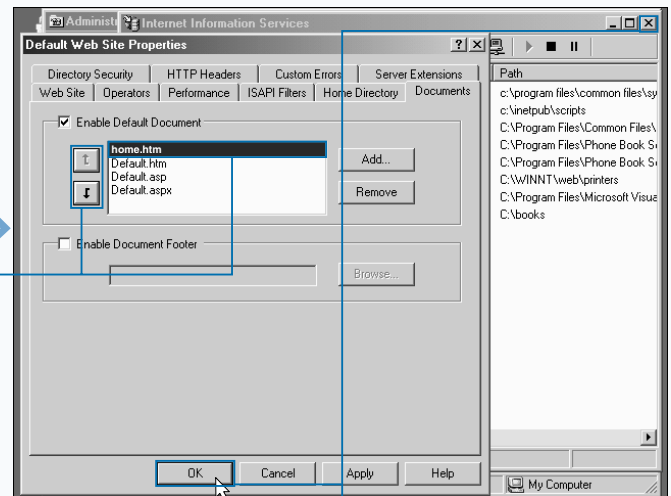
```
<bold>Copyright 2001</bold>
```



**8** Click Add to open the Add Default Document dialog box.

**9** Type the name of the new default document.

**10** Click OK to add the new default document to your list.



**11** The new default document is added to the list.

**11** Click an arrow button (↑ or ↓) to move the document up or down in the list.

**12** Click OK to confirm your changes and close the dialog box.

*Note: If the Inheritance Override dialog box appears, click Cancel to close the dialog box.*

**13** Click [X] to close the IIS window.

# CHANGE LOG FILE PROPERTIES FOR IIS

Internet Information Server log files contain information about requests made to your Web server. At the time of their request, you can have IIS log a number of different details about the request. When the Enable Logging option is checked in your Web Site tab of the Properties window for your Web site, IIS logs requests.

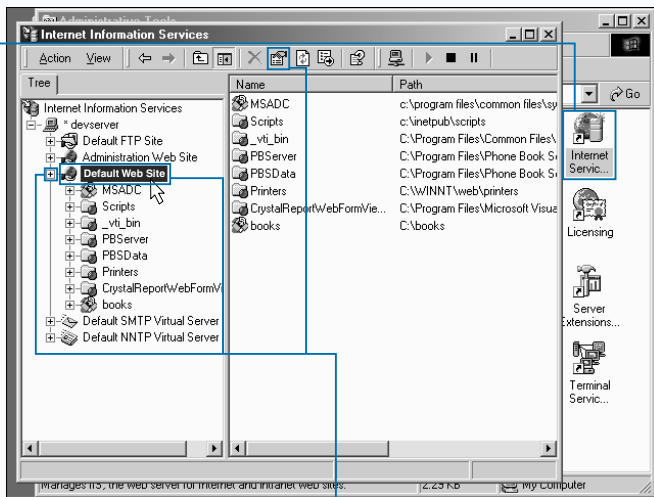
You can choose the format in which you want the log files to be saved. The default format of W3C extended format should work for most cases, unless you have a program that requires some other format.

You can specify how often you want log files to be created for your Web site. For busier sites, hourly is

appropriate. You can choose to put your log file information in the same log file by choosing Unlimited file size. You can also specify the location of the log file in the Web server. The default location of the log file is in the `System32\Logfiles` directory in the Windows operating system directory.

You should only use logging when it is necessary. Logging can have an impact on site performance due to its need to write to the file system. It can also be an issue if you are not removing old log files. You do not want your log files to fill up your hard drive.

## CHANGE LOG FILE PROPERTIES FOR IIS



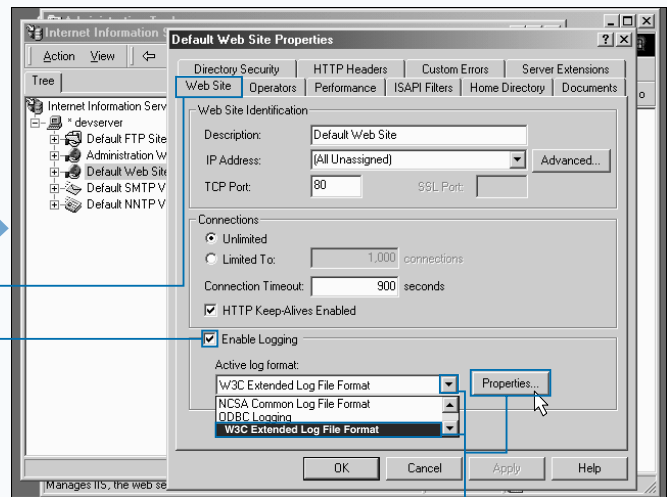
**1** In the Control Panel, double-click Administrative Tools to open the Administrative Tools window.

**2** Double-click Internet Services Manager to open the Internet Information Services window.

**3** Click **+** to expand the list of Web sites on the Web server (**+** changes to **-**).

**4** Click the Web site that you want to change.

**5** Click the Properties button (ⓘ).



**6** The Default Web Site Properties dialog box appears.

**7** Click  next to Enable Logging to enable the Web site to log information ( changes to ).

**8** Click **▼** to open the Active log format menu.

**9** Click to select the log file format that you want to use.

**10** Click Properties to open the Extended Logging Properties dialog box.

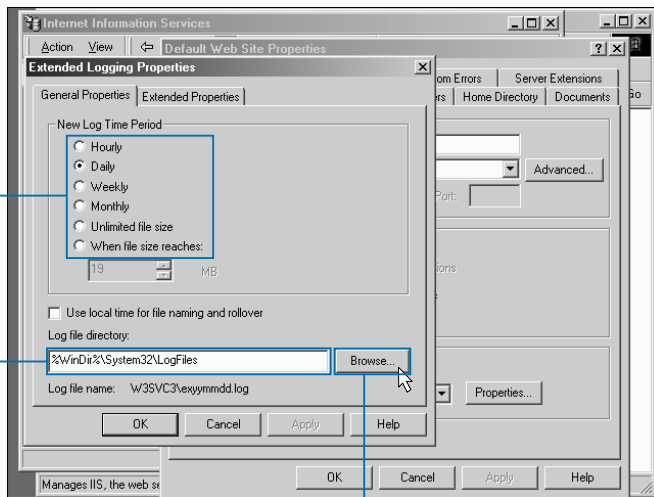
*Note: This option maybe enabled by default.*

**Extra**

You can select extended properties for assistance when troubleshooting. Click the Extended Properties tab and then click each desired extended logging option.

**Log File Example:**

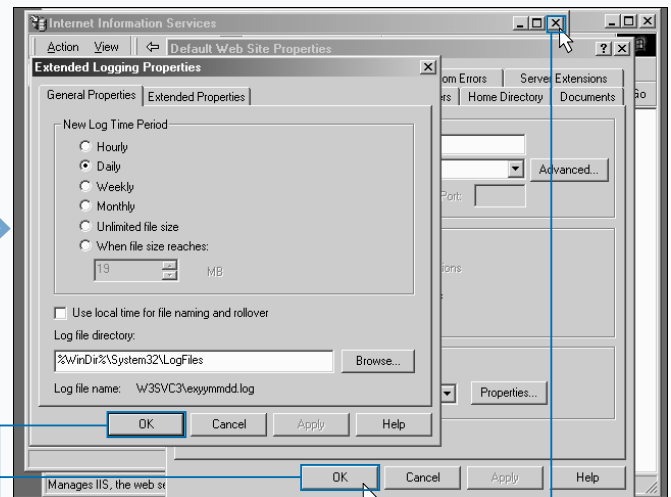
```
2001-04-20 15:26:22 127.0.0.1 - 127.0.0.1 80 GET /quickstart/aspplus/Default.aspx - 200 Mozilla/4.0+(compatible;+MSIE6.0b;+Windows+NT+5.0;+COM++1.0.2615)
2001-04-20 15:26:22 127.0.0.1 - 127.0.0.1 80 GET /quickstart/aspplus/images/aspplus2.gif - 304 Mozilla/4.0+(compatible;+MSIE6.0b;+Windows+NT+5.0;+COM++1.0.2615)
2001-04-20 15:26:22 127.0.0.1 - 127.0.0.1 80 GET /quickstart/aspplus/doc/toolbar.aspx - 200 Mozilla/4.0+(compatible;+MSIE6.0b;+Windows+NT+5.0;+COM++1.0.2615)
2001-04-20 15:26:23 127.0.0.1 - 127.0.0.1 80 GET /quickstart/aspplus/doc/quickstart.aspx - 200 Mozilla/4.0+(compatible;+MSIE6.0b;+Windows+NT+5.0;+COM++1.0.2615)
2001-04-20 15:32:10 127.0.0.1 - 127.0.0.1 80 GET /quickstart/aspplus/doc/whatisasp.aspx - 200 Mozilla/4.0+(compatible;+MSIE6.0b;+Windows+NT+5.0;+COM++1.0.2615)
```



**11** Click  to select the time period that you want to use to create new log files ( changes to .

**12** Type the path of the directory where you want to store log files.

**13** You can also click Browse to select a directory on the computer.



**13** Click OK to confirm the log file properties that you specify.

**14** Click OK to close the Properties dialog box.

**15** Click  to close the Internet Information Services window.

# STOP, START, OR PAUSE A WEB SITE

A Web site sometimes needs to be stopped, such as when you need to perform file maintenance, backups, or virus checks on the site. Stopping a Web site from running causes an immediate interruption of service for all users accessing the site. Any activity being performed by the Web site, such as processing an ASP.NET page or creating a file, is stopped immediately.

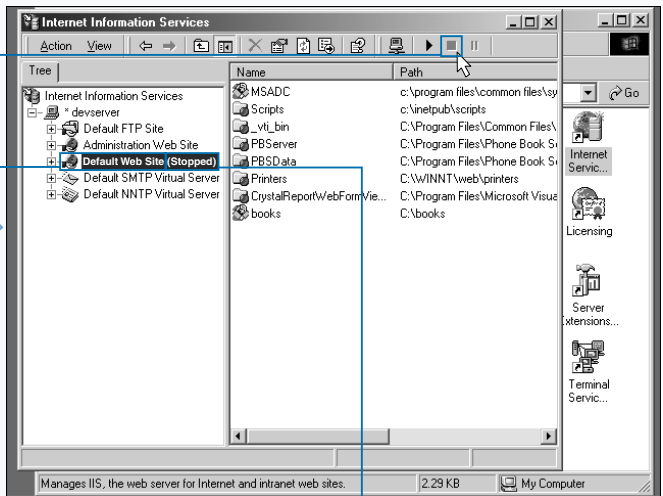
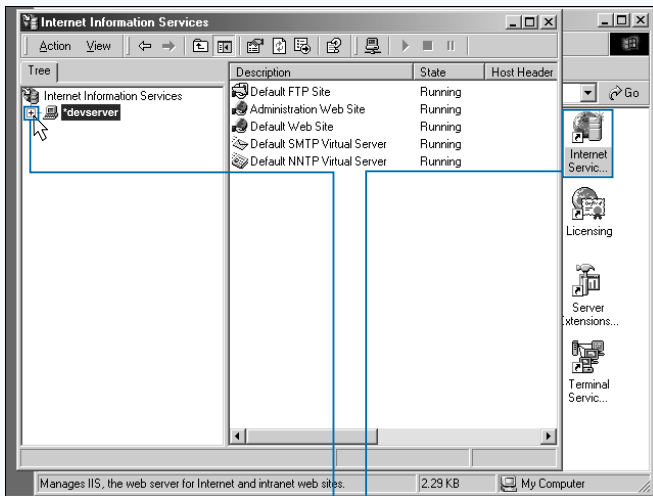
Some Web site configuration tasks can be performed while the site is running, but do not take effect until the site is restarted. You can stop and then start the site to apply the changes. Starting a Web site you

previously stopped also enables users to once again access information on the site.

Web sites can also be paused. Pausing a Web site does not stop the site from completing any activities that are in progress, but it prevents any new activity on the Web site. For busy Web sites, it is common for Web site administrators to first pause the Web site and then wait until all activity has ceased before stopping the Web site.

With ASP.NET applications, you should not have many reasons for stopping or pausing the Web server.

## STOP, START, OR PAUSE A WEB SITE



### STOP A WEB SITE

**1** In the Control Panel, double-click Administrative Tools to open the Administrative Tools window.

**2** Double-click Internet Services Manager to open the Internet Information Services window.

**3** Click to expand the list of Web sites on the Web server ( changes to ).

**4** To stop a Web site, click the Web site.

**5** Click the Stop button ().

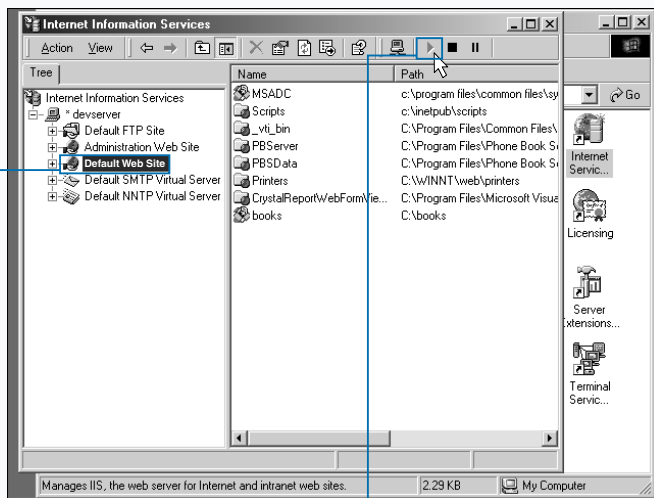
The Web site stops.



**Extra**

You can stop and start a Web server using the `iisreset` command. Using this command to stop or start a Web server will stop or start all the Web sites on the server. At the Command Prompt on the Web server, type `iisreset /` followed by the action you want to perform.

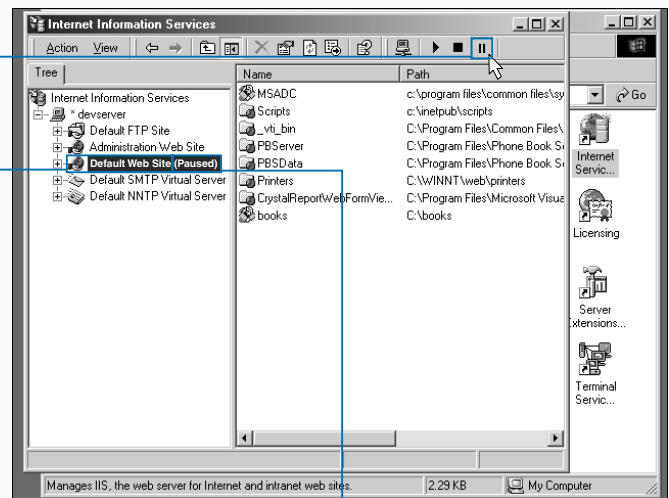
COMMAND	DESCRIPTION
<code>iisreset /restart</code>	Stop and then restart the Web server.
<code>iisreset /start</code>	Start the Web server.
<code>iisreset /stop</code>	Stop the Web server.
<code>iisreset /reboot</code>	Restart the computer.
<code>iisreset /rebootonerror</code>	Restart the computer if an error occurs while stopping, starting, or pausing the Web server.
<code>iisreset /status</code>	Stop and then restart the Web server.
<code>iisreset /?</code>	Display information about the <code>iisreset</code> command.

**START A WEB SITE**

**1** To start a Web site, click the Web site.

**2** Click the Start button (▶).

The Web site restarts.

**PAUSE A WEB SITE**

**1** To pause a Web site, click the Web site.

**2** Click the Pause button (▬).

The Web site pauses.

To resume running the Web site, click the Pause button again.

# BROWSE YOUR DEFAULT WEB SITE

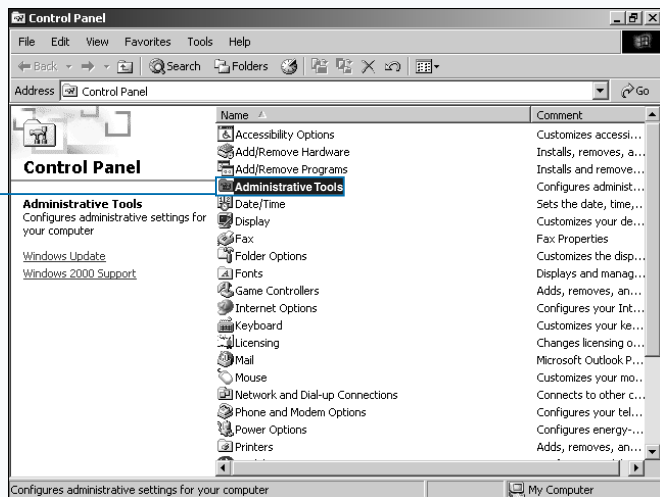
You can use the Internet Services Manager (ISM) Application to open pages into a browser for files that are on your Web site. You can open this tool directly from the Start menu or use the Snap-in console that is available in the Computer Management Console Application.

This administrative tool enables you to configure your Web site and navigate through all of the content on the site. You also can use the tool to open the Web site locally into your browser. You can also use the

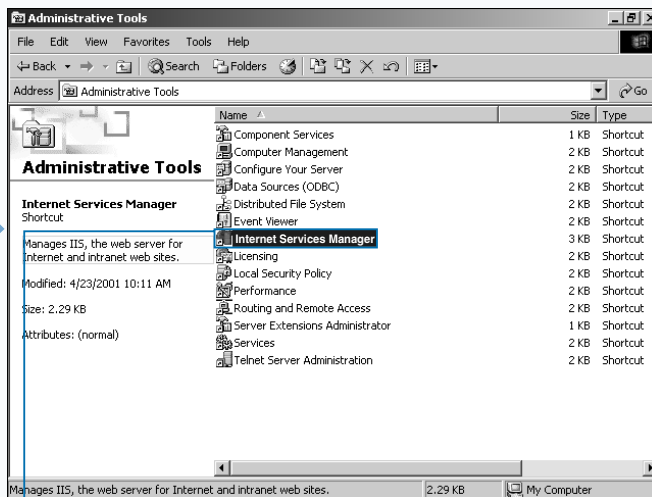
ISM tool to go to a specific directory or page in your Web site and browse from there. IIS Admin gives you an Explorer-like view of your Web site so it is very intuitive for you to go to any specific area of your site. Having this tool enables you to easily locate any page in your site and view it in a browser, versus having to type the entire URL.

If you choose a Web or directory to browse, the default document will appear. See page 14 for more information on setting a default document.

## BROWSE YOUR DEFAULT WEB SITE



**1** In the Control Panel, double-click Administrative Tools to open the Administrative Tools window.



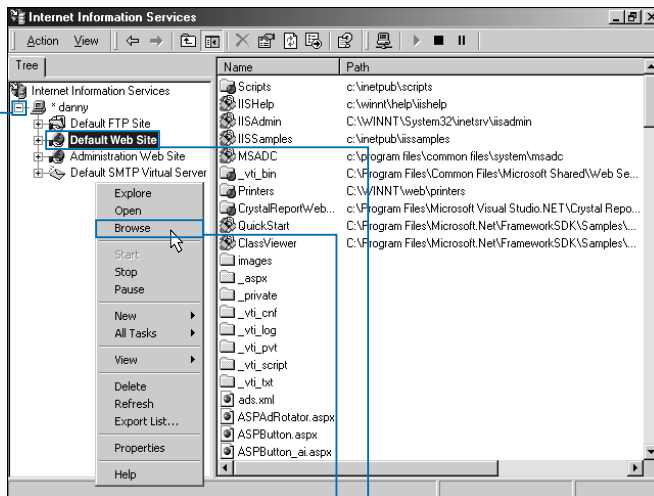
**2** Double-click Internet Services Manager to open the Internet Information Services window.

**Extra**

The Internet Services Manager is useful for browsing your Web site, but if you are browsing the same page multiple times, then you should look at some other alternatives. One alternative is to add the URL to your list of favorites in your browser.

You can also launch a page from a shortcut on your desktop. You can create a shortcut by right-mouse clicking on your desktop and choosing the option off of the pop-up menu (New ⇨ Shortcut). This brings up a dialog box that takes you through a wizard to configure the shortcut.

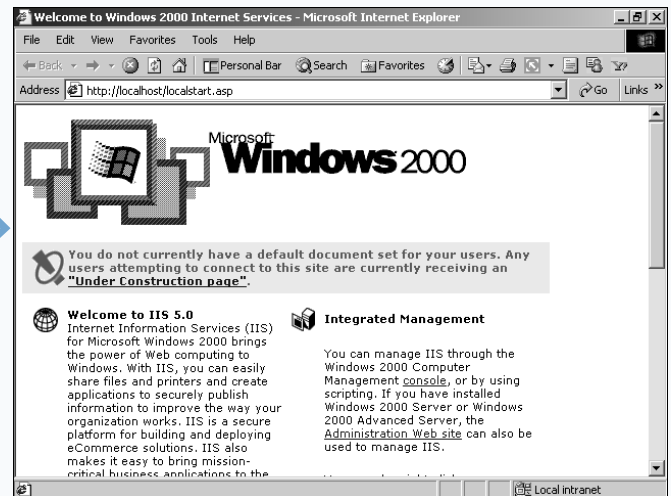
You can also browse your site with other tools, like your development environment tool. One example is using Visual Studio .NET (VS.NET). VS.NET has a project explorer built into the development environment. From the project explorer you can launch any file you choose. You even have the option of choosing which browser to use.



**3** Click to expand the list of Web sites on the Web server ( changes to ).

**4** Right-click Default Web Site to open the pop-up menu.

**5** Click Browse.



■ The Default Web site appears.

*Note: The IIS 5.0 Documentation opens.*

# EXPLORE A WEB SITE

You can use the Internet Services Manager (ISM) Application to explore content on your Web site in Windows Explorer. You can open this tool directly from the Start menu or use the snap-in console that is available in the Computer Management Console Application.

This administrative tool enables you to configure your Web site and navigate through all of the content on the site. The tool gives you a Windows Explorer–like interface for navigating through the hierarchy of the site. You have the option to launch Windows Explorer from any directory of the site.

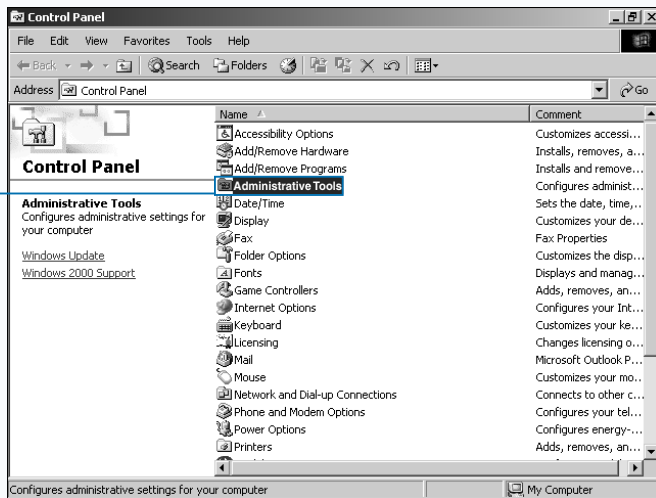
Being able to launch Windows Explorer from any location in the ISM tool is very convenient. The ISM

tool's interface displays a virtual structure of your Web site.

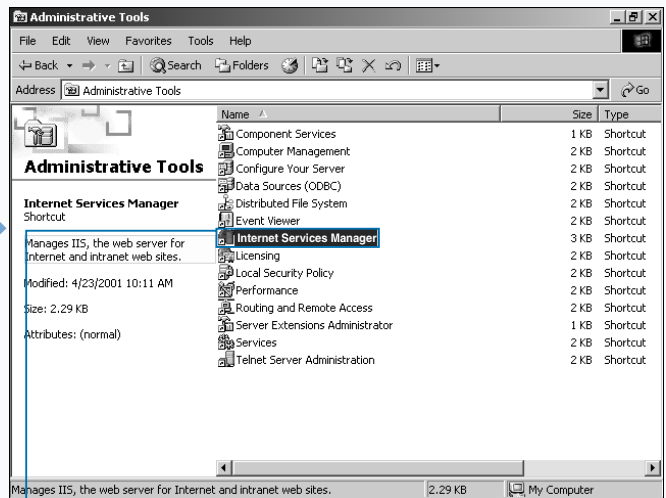
There are cases in which you want to access the physical files or directories in your Web site. These cases could be when you want to set file/directory permissions or open a file directly for editing.

Exploring your site through ISM is a useful feature when your site is composed of many virtual directories. With virtual directories, when you choose the Explore action from ISM, it loads up the directory to which the virtual directory is mapped. If the virtual directory is mapped to a share, the share appears in Windows Explorer.

## EXPLORE A WEB SITE



**1** In the Control Panel, double-click Administrative Tools to open the Administrative Tools window.

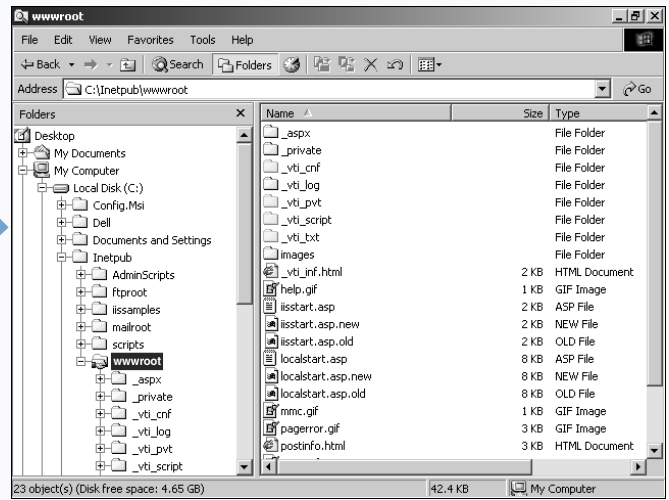
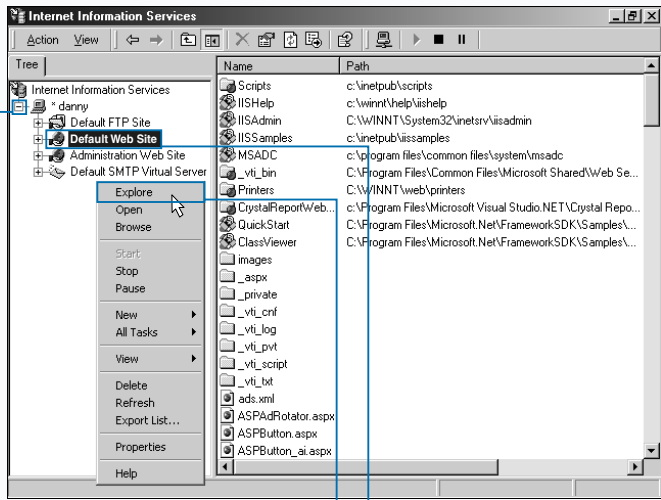


**2** Double-click Internet Services Manager to open the Internet Information Services window.

**Extra**

An alternative way to get to the Internet Services Manager is to open the Computer Management Console. This console can be reached by right-mouse clicking on the “My Computer” icon on the desktop and choosing Manage from the pop-up menu. You can also get to the Computer Management Console through the Control Panel.

The Computer Management Console is a very useful console when it comes to managing your Web site. You can connect to other machines on your network and manage resources on those machines that are used by your Web site. For example, you can maintain the shares that are exposed by a remote machine. For a share, you can give the share name, path, and security rights. Shares are sometimes used for virtual directories in your Web sites. See page 10 for information on how to create a virtual directory.



**3** Click **+** to expand the list of Web sites on the Web server (**+** changes to **-**).

**4** Right-click Default Web Site to open the pop-up menu.

**5** Click Explore.

The Windows Explorer displays the directory where the files for the Default Web site are located.

# OPEN A TEMPLATE FILE

You can use template files to shorten the time it takes to program your applications. Template files also help promote more consistent use of programming standards.

Template files are files with common code that can be used across an application. For example, this book uses a template file, `GenericTemplate.aspx`, as a starting point for creating an ASP.NET Web page.

There are many repetitive lines of code in creating an application. For example, Web Forms have several lines of code that are very similar page after page, with the exception of the content between the HTML opening and closing tags, which changes.

After you have built several Web applications, you can determine the classifications of pages and

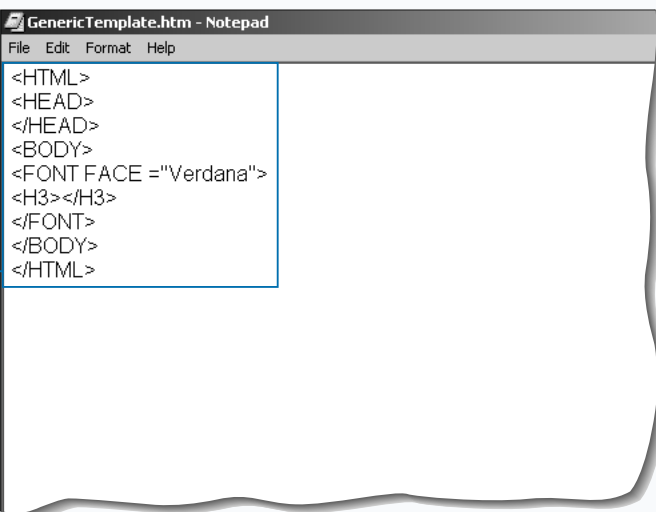
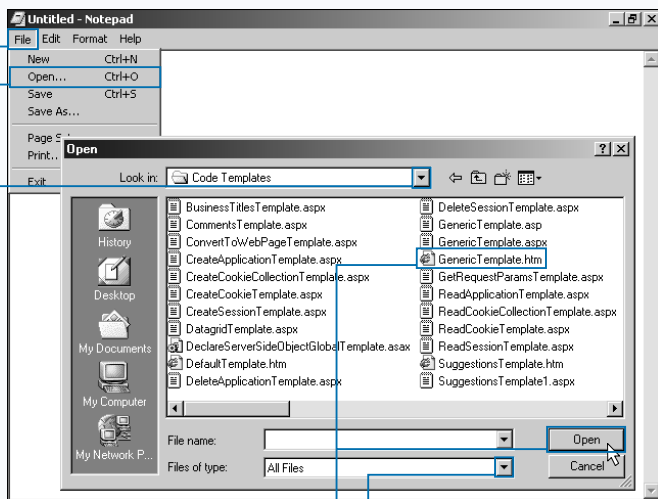
components that you build. When you see patterns of repetitive code, this is an opportunity to make a generic template that you can create for future use.

Templates are commonly used in this book to help you learn a task. By using the templates, you can save time by not having to create the basic HTML code for every page.

You can simply open the file directly from the CD with your text editor. Or, to avoid going to the CD drive, you can copy the files onto your hard drive.

After opening the template and making the necessary changes, you may want to save the file to the Web server for later requests.

## OPEN A TEMPLATE FILE



**1** Start your text editor.

**2** Click File ⇨ Open.

**3** Click to select the folder that contains your Code Templates.

**4** Click to select All Files from the drop-down list.

**5** Click to select a template.

**6** Click Open to open the template.

The template file loads into your text editor.

*Note:* If you installed the software from the CD, the path to the Code Templates directory is `C:\Program Files\Visual ASP.Net\Code Templates` plus the filename for the specific html template, such as `GenericTemplate.htm`.

*Note:* You can open the template from the CD at `D:\VISUALASPNET\Code Templates\`.

# SAVE A FILE TO THE DEFAULT WEB SITE

**Y**ou can save your files to the Default Web site so that you and others can request the file from the Web server.

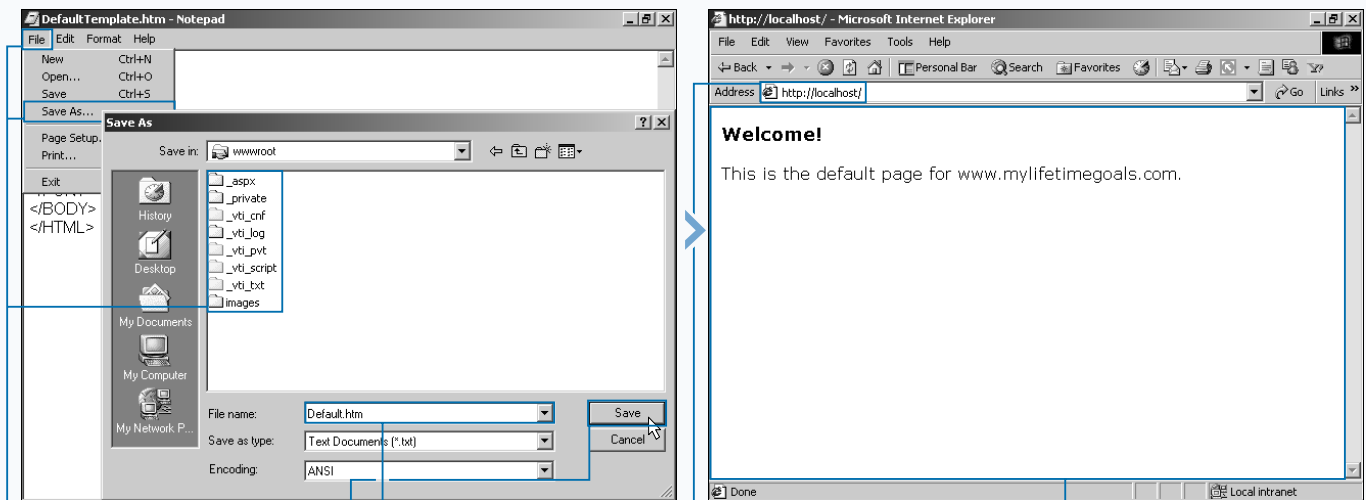
The Default Web site is a Web site that is on your Web server when you install Internet Information Server 5.0. If you have not mapped the Default Web site to another location, the default location for the Default Web site is `C:\InetPub\wwwroot`. That is the path you should use if you have not changed the default configuration.

If your Web server is on a different machine than the physical files of your Web site, you need to save the files on the file share that is designated for your Web

site, as opposed to a local drive on the Web server. To find the physical location of where your Web site files are stored, see page 22.

Common files that will make up your Web sites include Web Forms, Code-behind Pages, Web Services/Web Service Clients, components, and configuration files. Unlike ASP 3.0 sites that use components, ASP.NET applications are self-describing and do not require registry entries. Therefore, you can simply copy the application with `XCOPY` or `FTP`. Copy deployment works in most cases, but there are other configurations that take a few more steps to properly configure.

## SAVE A FILE TO THE DEFAULT WEB SITE



**1** Click File → Save As.

**2** Click to select the folder where you want to store your file.

■ If the folder where you want to store the files is not onscreen, click the Save in ▾ to select the folder.

**3** Type a name for the file.

**4** Click Save to save the Web page.

*Note: To enable the file to load automatically in the Default Web site, save the file to the root directory (C:\inetpub\wwwroot) as Default.htm.*

**5** Browse to the Default Web site at `http://localhost`.

■ The Web page appears in the Default Web site.

*Note: See page 20 for instructions on browsing the Default Web site.*

*Note: See page 14 for more information about the default document in IIS.*

# CREATE AN HTML PAGE

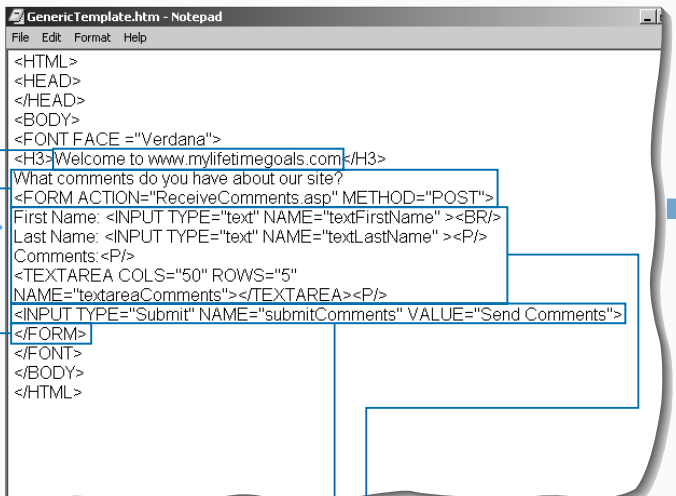
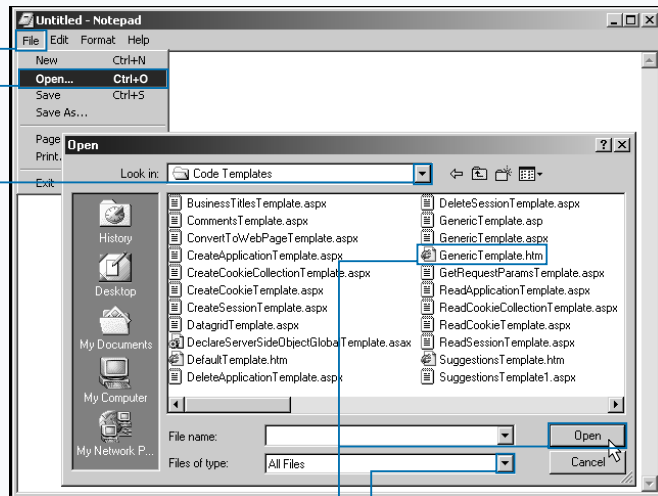
You can use a text editor to create a Web page. A Web page should, at a minimum, have the `<HTML>` and the `<BODY>` tags in it. The tags in your HTML page give your Web browser specific instructions for displaying the page to the user. Most tags have an opening and a closing tag that affect the text between the tags. The closing tag should have a forward slash (/) in it. It is common to type tags in uppercase letters to make the tags stand out from the text in the Web page.

The `<HTML>` tag identifies a document as a Web page. You must place the HTML markup that you want to appear in the Web browser between the `<BODY>` tags.

You can create an HTML page with a form on it to gather input from a user. A form can be placed anywhere between the `<BODY>` tags in an HTML page. The action attribute in the `<FORM>` tag tells which page should process the form when the user submits the form.

Forms are the best way to collect input from a user. You will consistently use forms if you are building an interactive site. HTML forms are a big part of how the ASP.NET has implemented its Framework, as you will see with Web Forms.

## CREATE AN HTML PAGE



- 1 Start your text editor.
- 2 Click File ⇨ Open.
- 3 Click ▾ to select the folder that contains your Code Templates.

- 4 Click ▾ to select All Files from the drop-down list.
- 5 Click to select a template.
- 6 Click Open to open the template.

- 7 Type a description of the page between the `<H3>` tags.
- 8 Position the cursor insertion point between the `<BODY>` tags and create a form for collecting information about user comments.

- 9 Add additional HTML controls to the form to collect information.
- 10 Add a Submit button to the form.

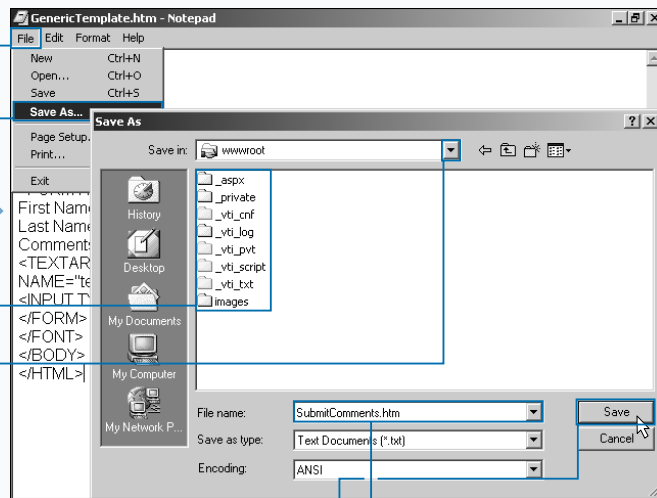


## Extra

Several elements on an HTML form enable you to collect and/or display information. An example of an element is a `textbox`. For each element that you have, you can have a number of attributes. An example of an attribute for the `textbox` is the `maxLength` property, which tells the Web browser the maximum number of characters that the user should be able to type into the `textbox`.

To build a form, here are some examples of commonly used form elements:

FORM ELEMENTS	DESCRIPTION
Text Box	A text box enables users to enter a single line of text, such as a First Name.
Text Area	The <code>TEXTAREA</code> element displays a large text area that enables users to enter several lines or paragraphs of text.
Password Box	A password box enables users to enter text into a text box while masking what they are typing in with asterisks.
Drop-Down List	The <code>SELECT</code> element displays a drop-down list that enables users to select an option from a list of several options.
Check Box	Check boxes enable users to select one or more options.
Radio Button	Radio buttons enable users to select only one of several options.
Submit Button	A submit button enables users to send data in the form to the ASP page that will process the data.



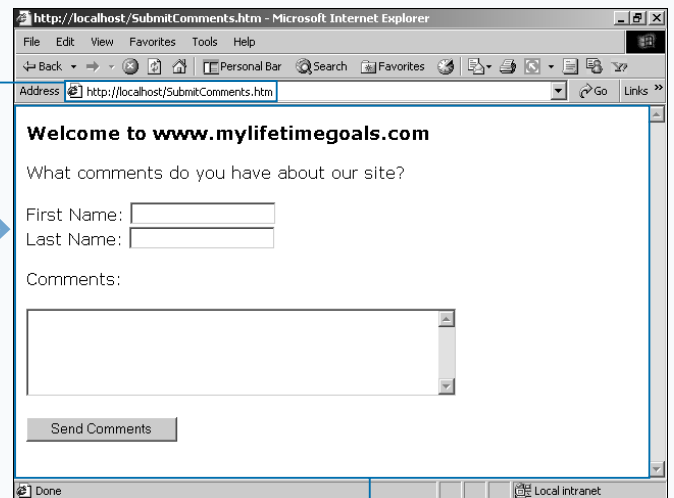
11 Click File → Save As.

12 Click to select the Default Web Site folder where you want to store your file.

■ If the Default Web Site folder is not onscreen, click the Save in ▾ to select the folder.

13 Type **SubmitComments.htm** as your filename.

14 Click Save to save and close the dialog box.



15 Browse to the saved Web page at `http://localhost/SubmitComments.htm`.

*Note:* See page 20 for instructions on browsing the default Web site.

■ The Web page appears with the form on it.

*Note:* If you click the Send Comments button on this form, a **File not Found error** will appear onscreen because you have not created the page.

# CREATE AN ASP PAGE

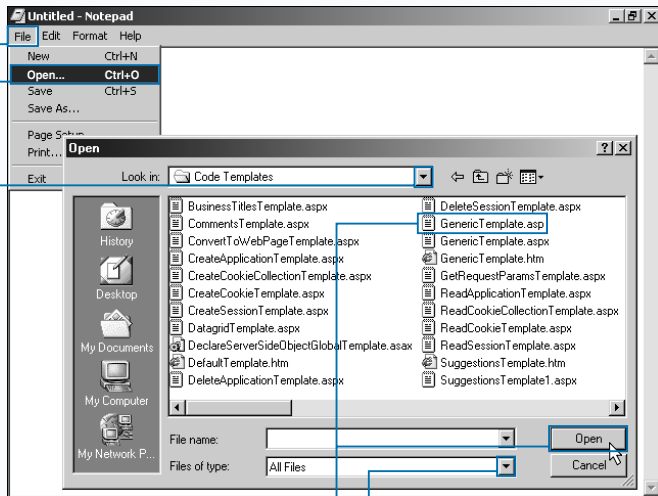
You can create an ASP 3.0 page to process a form that is submitted from an HTML page. ASP 3.0 pages and ASP.NET pages can coexist on the same Web site. Saving your ASP 3.0 pages with an .asp extension allows the Web server to determine how to properly process the file.

Adding ASP code to an HTML page enables you to create dynamic, interactive Web pages. ASP code is inserted into HTML code using *code declaration blocks*. Code declaration blocks are defined using `<script>` tags that contain a RUNAT attribute value set to "server." You can use `<%` and `%>` as a shorthand replacement for declaring server-side

code. The server-side code blocks tell the Web server where the ASP code begins and ends. This tells the Web server which areas need to be processed before being sent to the user's Web browser.

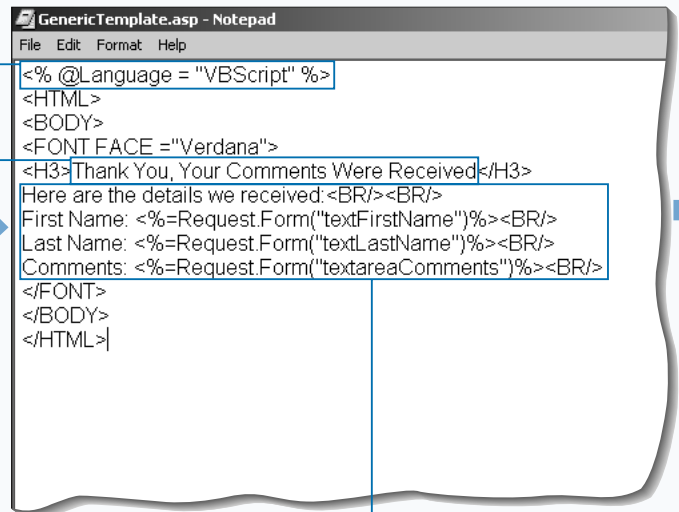
You can use the ASP 3.0 object model to interact with Web servers' requests and responses to users. The `Request.Form` statement enables you to access information passed by a form to a Web server. The `Response.Write` statement enables you to manipulate the response to a user's request. This statement enables you to modify the HTML sent in the response dynamically, enabling you to customize responses to user requests.

## CREATE AN ASP PAGE



- 1 Start your text editor.
- 2 Click File ⇨ Open.
- 3 Click ▾ to select the folder that contains your Code Templates.

- 4 Click ▾ to select All Files from the drop-down list.
- 5 Click to select a template.
- 6 Click Open to open the template.



- 7 Declare the scripting language used for the page.
- 8 Type a description of the page between the `<H3>` tags and press Enter.

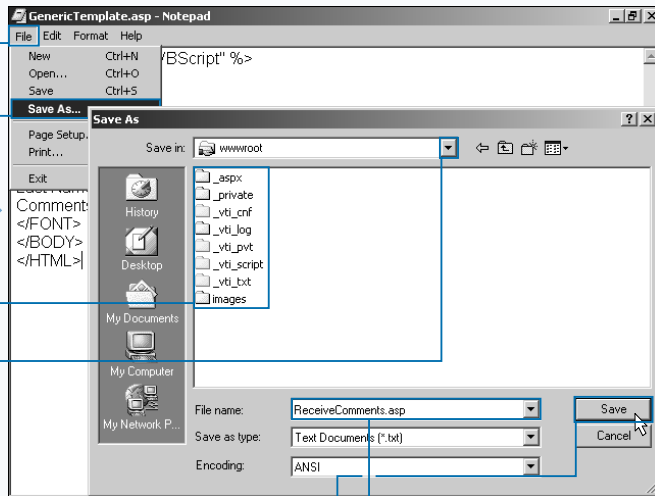
- 9 Type the HTML and ASP code to display what the user fills in on the form.

**Extra**

With proper use of the ASP 3.0 object model, you can generically display all of the information that was posted by any HTML form. The following code, written in VBScript, will display all of the form variables submitted.

**Example:**

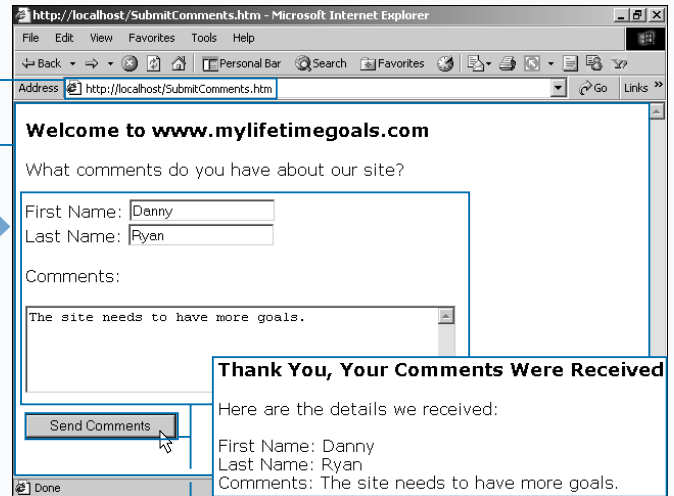
```
<P><STRONG>Form Collection</STRONG> </P>
<P>
<%
For Each Item in Request.Form
  For intLoop = 1 to Request.Form(Item).Count
    Response.Write Item & " = " &
      Request.Form(Item)(intLoop)
  %>
<br>
<% Next
  Next %>
</P>
```



- 10** Click File → Save As.
- 11** Click to select the Default Web Site folder where you want to store your file.
- If the Default Web Site folder is not onscreen, click the Save in  to select the folder.

- 12** Type **ReceiveComments.asp** as your filename.
- 13** Click Save to save and close the dialog box.

*Note: You will need to copy SubmitComments.htm to C:\inetpub\wwwroot\ from the installation directory or the CD.*



- 14** Browse to the saved Web page at `http://localhost/SubmitComments.htm`.
- Note: See page 20 for instructions on browsing the default Web site.*
- The Web page appears with the form on it.

- 15** Type your information into the fields on the form.
- 16** Click the Send Comments button.
- The Web page echoes back the information that you filled in on the form.

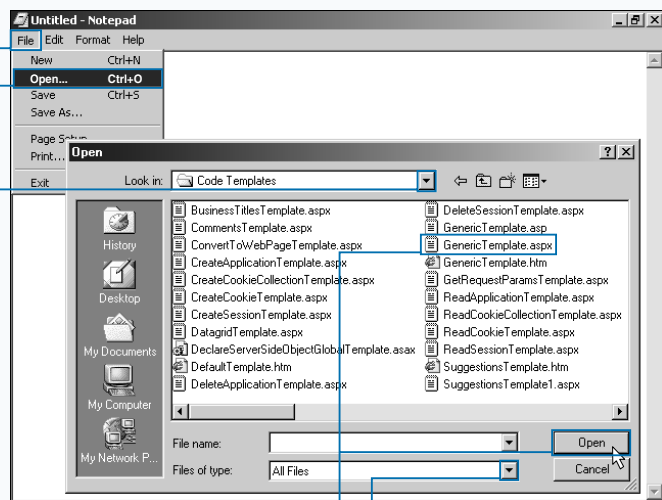
# CREATE AN ASP.NET WEB PAGE

You can create an ASP.NET Web Form that provides the HTML to render the user interface and processes requests from a user. You can do this by having the page submit to itself. To start the ASP.NET page, declare what language you will use on the page. Then add the standard HTML to the page, including the `<HTML>` and the `<BODY>` tags.

When you are ready to create the form for the page, you need to use the correct tags to do this. However, with ASP.NET, you must set up both the form and the controls on the form to run server-side. You can do this by setting the `RUNAT` attribute equal to `Server`. For more information about server-side controls, see Chapter 5, Introduction to Server-side Controls.

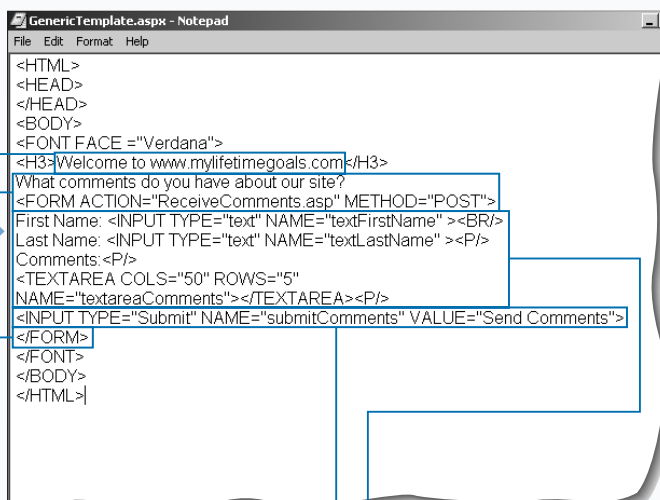
With ASP.NET Framework, the model for processing and responding to user requests is to use the same Web file(s) for collecting user input and displaying results. This model could be implemented with ASP 3.0, but you would typically see a separate page for collecting input and displaying results of submitted input. This model of sending requests to the same page that collects user input is called *postback*. Postbacks are an integral part of the event-handling model that is available in ASP.NET Web Forms.

## CREATE AN ASP.NET WEB PAGE



- 1 Start your text editor.
- 2 Click File ⇨ Open.
- 3 Click ▾ to select the folder that contains your Code Templates.

- 4 Click ▾ to select All Files from the drop-down list.
- 5 Click to select a template.
- 6 Click Open to open the template.



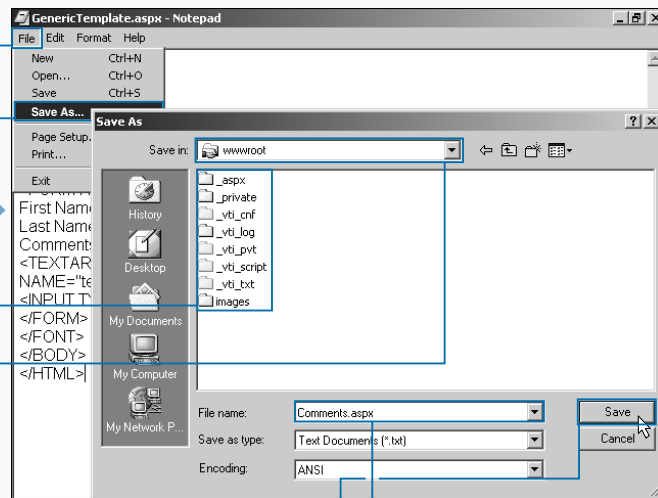
- 7 Type a description of the page between the `<H3>` tags.
- 8 Click between the `<BODY>` tags and create a form for collecting information about user comments.

- 9 Add additional HTML controls to the form to collect information.
- 10 Add a Submit button to the form.

**Extra**

Creating a Web Form that submits to itself, postback, is a common Web development technique. When you configure your Web Forms to postback, you will need a placeholder for displaying information when the page comes back to the user. You can put an empty `SPAN` tag into your Web Form as a placeholder for displaying results back to the user.

With ASP.NET Web Forms, you can either have your server-side code imbedded in the Web Form, extension is `.aspx`, or place the code in a code-behind page, in the case of C# the extension is `.cs`. The concept of code-behind pages is a new feature that was not available in ASP 3.0. Code-behind pages enable you to separate code from presentation (HTML). This is a cleaner programming model that is closer to what is available to event-driven programming tools that programmers have had for years.



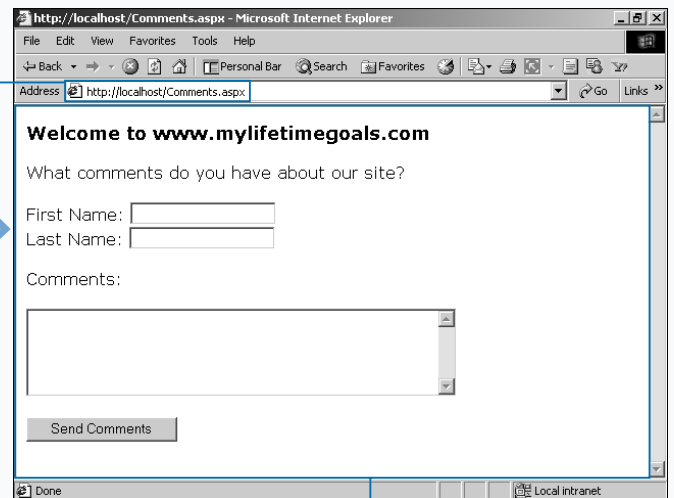
**11** Click File → Save As.

**12** Click to select the Default Web Site folder where you want to store your file.

■ If the Default Web Site folder is not onscreen, click the Save in ▾ to select the folder.

**13** Type **Comments.aspx** as your filename.

**14** Click Save to save and close the dialog box.



**15** Browse to the saved Web page at `http://localhost/Comments.aspx`.

*Note:* See page 20 for instructions on browsing the default Web site.

■ The Web page appears with the form on it.

*Note:* If you click the Send Comments button on this form, nothing significant should happen because no code has been created for when the page submits the form to itself.

# ADD AN EVENT HANDLER TO AN ASP.NET PAGE

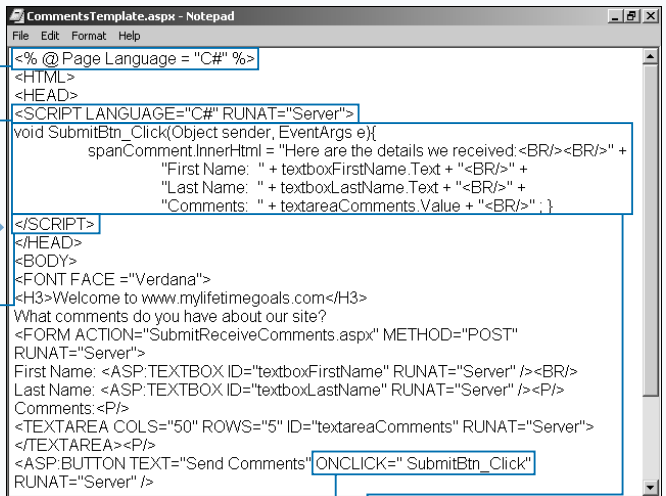
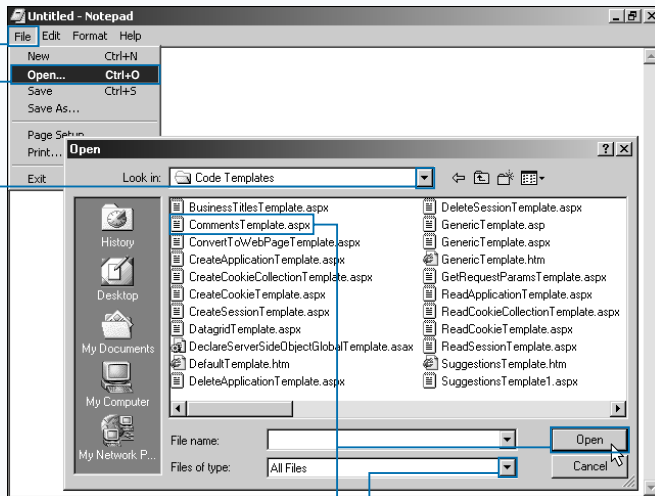
You can add code to your ASP.NET pages to respond to your user's interaction, for example, clicking a button or an image. In Web development, you are presented with a challenge when programming for events. If you want to process events on the Web server, you need to forward the event captured by the browser to the Web server along with the state of the page.

Processing user interactions to elements on your Web page (events) on the Web server is simplified for you in ASP.NET. To configure this, you need to do a few things to properly configure your Web Form. First,

make sure your HTML form has the `RUNAT` attribute set to `Server`. Next you need to add an additional attribute to the element that is capturing the user interaction. This attribute determines what function is called when the event occurs. For example, you can add an `OnClick` attribute to the button that calls a function to process the form. From within the function, you can set the `<SPAN>` placeholder's `InnerText` property to display some HTML based on results of processing the event.

Note that this is only one of several ways to configure events for your Web Forms.

## ADD AN EVENT HANDLER TO AN ASP.NET PAGE



- 1 Start your text editor.
- 2 Click File ⇨ Open.
- 3 Click ▾ to select the folder that contains your Code Templates.

- 4 Click ▾ to select All Files from the drop-down list.
- 5 Click to select a template.
- 6 Click Open to open the template.

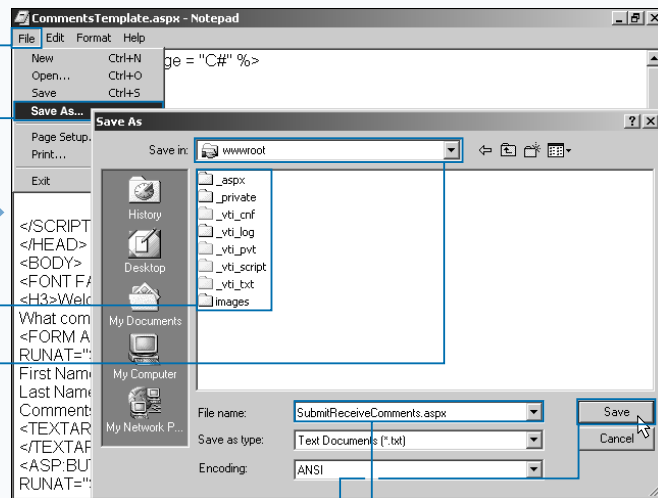
- 7 Declare the language used for the page.
- 8 Declare a script block to run server-side.

- 9 Create an event handler.
- 10 Add the ONCLICK event handler to the form and call the event created.

**Extra**

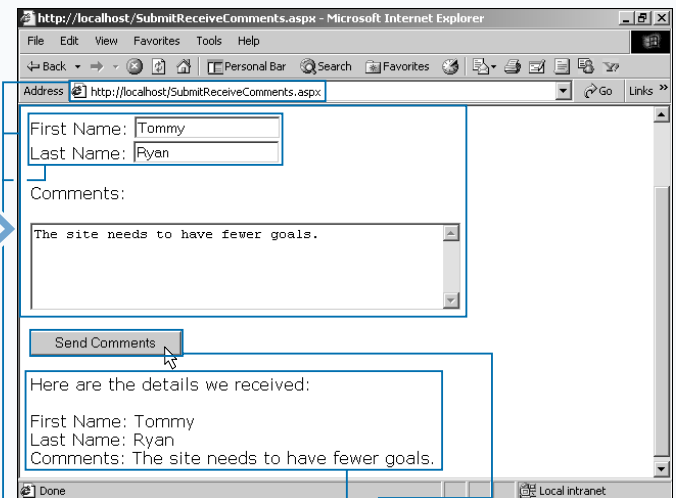
The Web Forms architecture gives you the ability to run server-side code when client-side events are fired. This is how you can run C# code when the button is clicked. The Web Forms framework handles virtually all of the mechanics of capturing, transmitting, and interpreting the event and handling it appropriately on the server.

Instead of responding to events on the server, you can write client-side script that handles events on your Web page. These scripts are interpreted by the client's browser. Because users can have one of many browsers, you are not assured that the event-handling code will run properly. Also you are not assured that the user will access your page with a browser that can interpret script. There are programmatic ways to address these issues, but you will find that server-side code is a more reliable way to process events.



- 11 Click File → Save As.
- 12 Click to select the Default Web Site folder where you want to store your file.
- 13 If the Default Web Site folder is not onscreen, click the Save in ▾ to select the folder.

- 13 Type **SubmitReceiveComments.aspx** as your filename.
- 14 Click Save to save and close the dialog box.



- 15 Browse to the saved Web page at <http://localhost/SubmitReceiveComments.aspx>.
- 16 The Web page appears with the form on it.
- 16 Type your information into the fields on the form.

- 17 Click the Send Comments button.
- 17 The Web page echoes back the information that you filled in on the form.



# WRITE YOUR FIRST C# APPLICATION

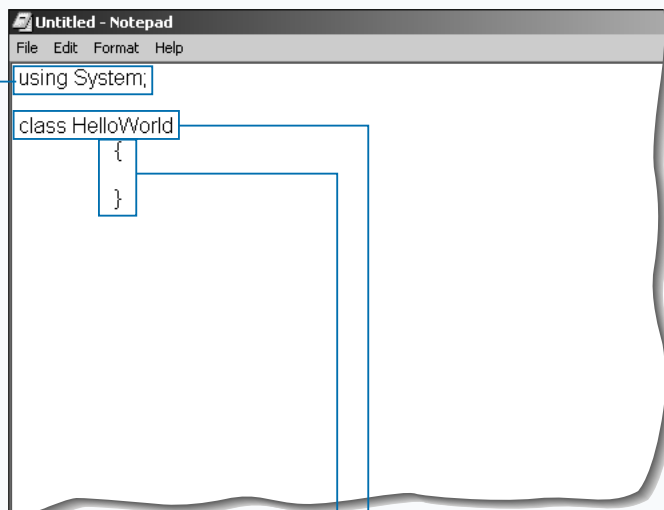
The C# (pronounced C sharp) language is a valuable asset in the .NET Framework. You can use C# to create standalone executables or to create dynamic content on a Web form.

If you are new to the C# language, you may want to start applying it by creating the simplest C# application possible. There are many types of applications that can be built with the C# language. Some of the more common ones that you can create are Windows applications, Web services, ASP.NET applications, and console applications. If you want the simplest application, a console application is a good choice.

Starting with the C# language, you can create a standard “Hello World” console application. To write your first application, you need a text editor, like Notepad, to generate the source code. The file type that typically holds C# code is a *class file*. A C# class is a text file saved as a \*.cs file. You can compile this class with the C# compiler (*csc.exe*) at the command prompt which creates an executable (\*.exe) file.

In C# applications, you can use *Namespace aliases* to easily leverage the .NET Framework classes. Namespace aliases are used to reference classes in the .NET Framework.

## WRITE YOUR FIRST C# APPLICATION



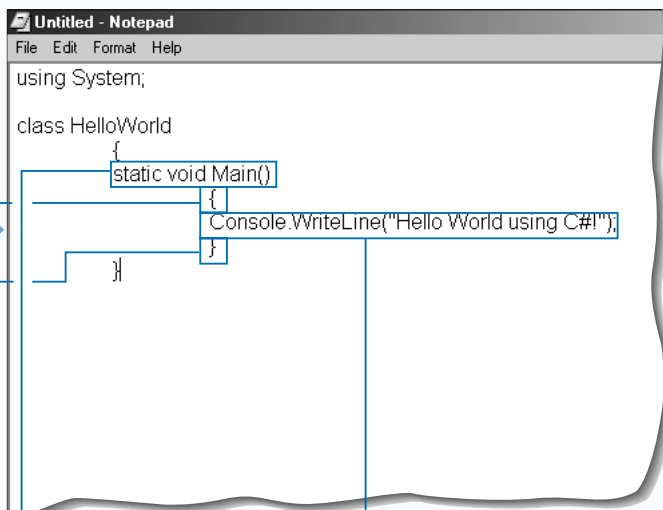
```

using System;

class HelloWorld
{
}
  
```

- 1 Open your text editor.
- 2 Type **using System;** to import the System namespace and press Enter.

- 3 Type the name of the class you want to create and press Enter.
- 4 Type {}, placing the opening and closing curly braces on separate lines, to set off the body of the class.



```

using System;

class HelloWorld
{
    static void Main()
    {
        Console.WriteLine("Hello World using C#!");
    }
}
  
```

- 5 Between the curly braces, type **static void Main()** and press Enter to create the Main function.

- 6 Type {}, placing the opening and closing curly braces on separate lines, to set off the body of the Main function.

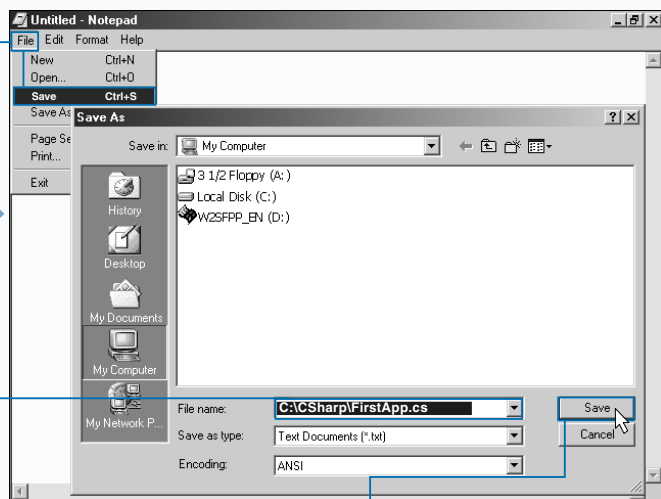
- 7 Between the curly braces, type in the code to print out "Hello World using C#!".

**Extra**

If you are familiar with object-oriented programming, you may know that a class defines the operations an object can perform, such as methods, events, or properties, and defines a value that holds the state of the object. Although a class generally includes both definition and implementation, it can have one or more members that have no implementation.

An instance of a .NET Framework class is an object. You can access an object's services by calling its methods and accessing its properties, events, and fields. Each language chooses its own syntax for creating instances of classes.

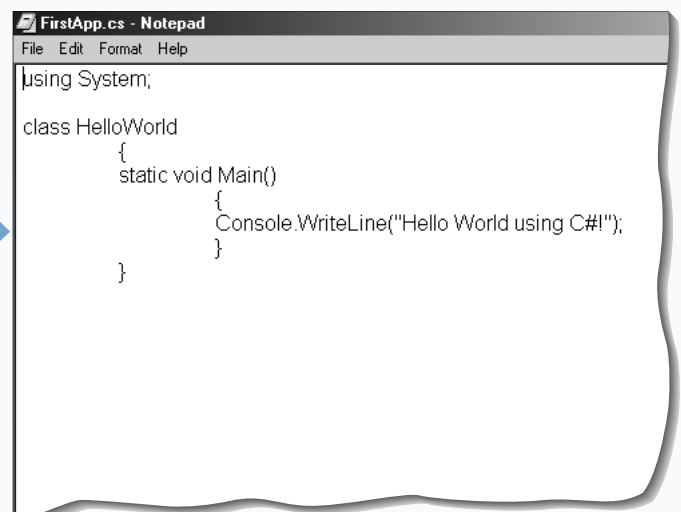
The C# language is case sensitive. This will be one of the first bugs you will find when compiling your C# application. To avoid problems with case sensitivity, you can use an editor and compiler that assists by being compliant with the case sensitivity.



**8** Click File → Save to open the Save As dialog box.

**9** Type a name for the file.

**10** Click Save.



■ The source file saves to the directory and can now be compiled.

*Note: You can save all of your console applications in a specific directory (example: C:\CSharp).*

# COMPILE A C# APPLICATION

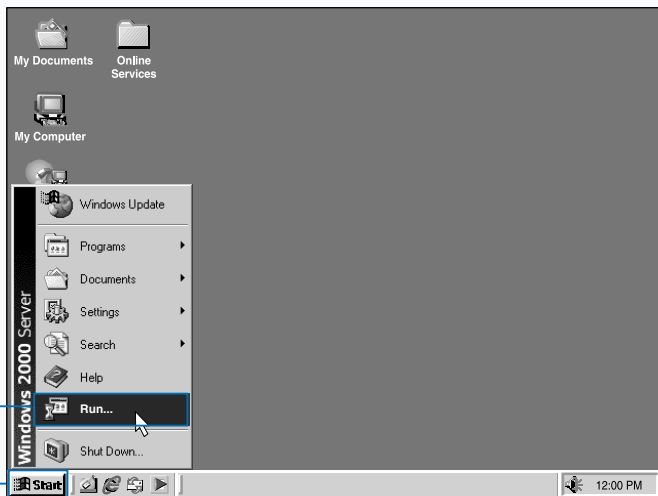
Once you have created your C# source file, you can compile the file into an executable program that you can run. A *compiler* is a utility program that takes a source file — a readable text file — and converts it into a executable file — a binary file that the operating system of the computer knows how to run.

To compile the application, open the command prompt and go to the directory where the source file is located (for example: `C:\Csharp`). You can use the `csc.exe` command to invoke the C# compiler. To

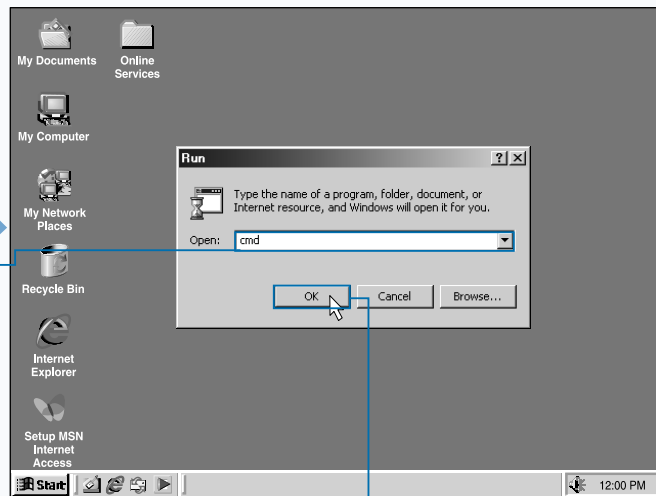
specify the source file that you want to compile, simply type a space and then the name of the file that you want to compile.

When this command is run, the source file is used by the C# compiler to create an executable program. If you created a console application, you can then run this executable program by typing the name of the file. A *console application* is an executable program that can be run from the command line. The C# compiler has many options that you can specify when compiling an application.

## COMPILE A C# APPLICATION



1 Click Start ⇨ Run.



2 The Run dialog box appears.

2 Type `cmd` in the Open field.

3 Click OK to open the command line window.

**Extra**

You can specify another filename for the executable application that you create. To do this, use the `/out` switch followed by a colon and type the name of the executable file that you want to create. For example, to compile the `FirstApp.cs` source file to an executable file named `HelloWorld.exe`, you would type `csc /out:HelloWorld.exe FirstApp.cs`

You can view all of the options for the C# compiler by typing `csc /?` at the command line. Because the options list is long and could scroll outside of the viewable window, type `csc /? | more` to see the options one screen at a time.

```

C:\WINNT\System32\cmd.exe
Microsoft Windows [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\Documents and Settings\Administrator>cd C:\CSharp

C:\CSharp>csc FirstApp.cs
Microsoft (R) Visual C# Compiler Version 7.00.9148 [CLR version
6151]
Copyright (C) Microsoft Corp 2000. All rights reserved.

C:\CSharp>FirstApp.exe
  
```

**4** Change directories to the source file location by using the `cd` command.

*Note: In this example, the command is `cd C:\CSharp`.*

**5** Compile the class at the command prompt using the `csc` command.

*Note: In this example, the command is `csc FirstApp.cs`.*

■ An executable file is created from the source file.

**6** Run the program by typing the name of the executable file and pressing Enter.

```

C:\WINNT\System32\cmd.exe
Microsoft Windows [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\Documents and Settings\Administrator>cd C:\CSharp

C:\CSharp>csc FirstApp.cs
Microsoft (R) Visual C# Compiler Version 7.00.9148 [CLR version
6151]
Copyright (C) Microsoft Corp 2000. All rights reserved.

C:\CSharp>FirstApp.exe
Hello World using C#!

C:\CSharp>
  
```

■ The program displays the Hello World using C#! message.

# FORMAT YOUR CODE

**W**ell-formatted code makes your code easier to read, maintain, and reuse. Formatting your code professionally is an important consideration in development. Before your development team starts coding, they should write a coding guidelines document. These guidelines should include all the standards that determine how your organization formats code professionally. If these guidelines are not set at the beginning, the code will look as though many individual developers, as opposed to a coordinated development group, created the code. Formatting code is a discipline that is carried out during the coding, not at the end of it. If developers wait until after the coding to assemble guidelines, the task will most likely not get done.

When formatting your code, be sure to put in white space, comments, and indents. Most developers have good structure to their code, but skip over the task of *commenting*, which is necessary for capturing the why, what, when, and how of their code. Commenting your code while producing it facilitates better communication within the development team and helps with the maintainability. If you do not comment when you code, you may end up not documenting the code. A common mistake for development teams is putting off commenting until the very last part of the project and then never getting back to complete the task.

## FORMAT YOUR CODE

```

Untitled - Notepad
File Edit Format Help
namespace FormatCode
{
// Reference System namespace classes
}
  
```

- 1 Open your text editor.
- 2 Type the name of the namespace you want to create and press Enter.

- 3 Type {}, placing the opening and closing curly braces on separate lines.
- 4 Between the curly braces, press Tab and type // to begin a single line comment, add the comment details, and then press Enter.

```

Untitled - Notepad
File Edit Format Help
namespace FormatCode
{
// Reference System namespace classes
using System;

/// <summary>
/// This Class is to demonstrate good
/// code formatting practices.
/// </summary>

class MainApp
{
}
}
  
```

- 5 Type **using System;** to import the System namespace and press Enter.
- 6 Add a documentation comment by typing /// followed by the comment.

- 7 Type the name of the class you want to create followed by {}, placing the opening and closing braces on separate lines.

**Extra**

To properly format code, you must keep several concepts in mind. Whitespace is where you control the density of your code; this whitespace includes blank lines. Also, consider the amount of code on each line, as well as indentation. Indention is a key part of giving structure to your code. You will need to know your language's proper syntax when formatting your code. The key parts of the formatting that are affected by syntax are how you show a line continuation, a line end, and the keywords used for comments. The line end is notated by the semicolon (;) and the line continuation is the carriage return and line feed (just strike the enter key in your editor). In C# you can comment a single line or multiple lines. A single line is notated by // and a multiple line is notated by /\* at the beginning of the first line and \*/ at the end of the last line.

```

Untitled - Notepad
File Edit Format Help

/// <summary>
/// This Class is to demonstrate good
/// code formatting practices.
/// </summary>

class MainApp
{
    /*
    * Author: Tommy Ryan
    * Date: 04/26/2001
    * Description: This function is the main entry
    * point to the application.
    *
    * Params: N/A
    * Return Values: N/A
    * Revisions: 04/26/2001 - Initial Revision
    */
    public static void Main()
    {
  
```

**8** To create a flower box, type / to specify a multiple line comment.

**9** Type in the details of the class.

**10** To end the flower box, type / to specify the end of a multiple line comment.

**11** Type **public static void Main()** followed by { }, placing the opening and closing curly braces on separate lines, to create the **Main** function.

```

Untitled - Notepad
File Edit Format Help

    * Author: Tommy Ryan
    * Date: 04/26/2001
    * Description: This function is the main entry
    * point to the application.
    *
    * Params: N/A
    * Return Values: N/A
    * Revisions: 04/26/2001 - Initial Revision
    */
    public static void Main()
    {
        // Declare a variable(s)
        string sCurrentDate;

        // Get the current date
        sCurrentDate = GetCurrentDate();

        // Write a line of text to the console
        Console.WriteLine("The current date "
            + "on the system in short date format is "
            + sCurrentDate);
    }
  
```

**12** Within the **Main** function, add a single line comment by typing //.

**13** Type the comment.

**14** Type the remaining code, adding comments on the line before the code.

# DECLARE A VARIABLE

**Y**ou can structure the way you store information with variables. When you declare variables, you set up locations in memory where your program can store the many values it needs to perform its task. Defining a variable enables you to assign an easy-to-remember name to a memory location.

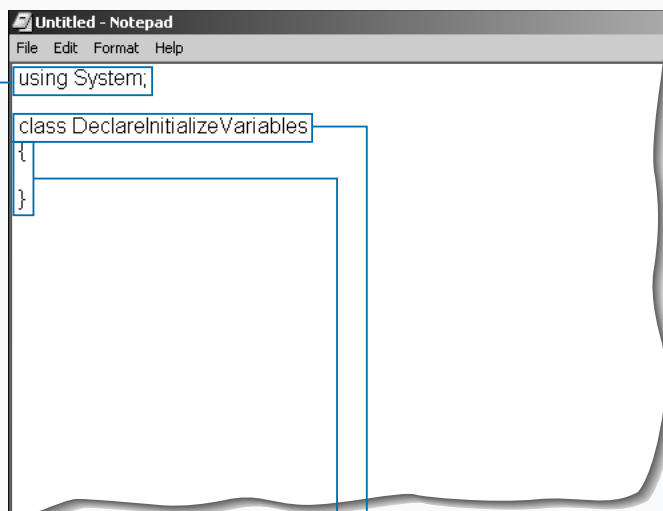
In the real world, you deal with many classifications of information. This information could be a date, money, a person's name, or an age. When you build applications that use this information, you will need to temporarily store these values for later use in the application.

Using variables in C# requires declaration and initialization. *Declaration* tells the application how to

allocate memory for the information you want to store. This is the first step in using variables and is required before you perform the initialization. *Initialization* is setting an initial value for the variable.

To declare a variable in C#, you will need to determine what type of information you want to store. The type of information you are storing will determine what the proper data type to use is. If you are storing someone's first name, you can use the string data type for its storage. If you are storing someone's age, you can use one of the integer data types.

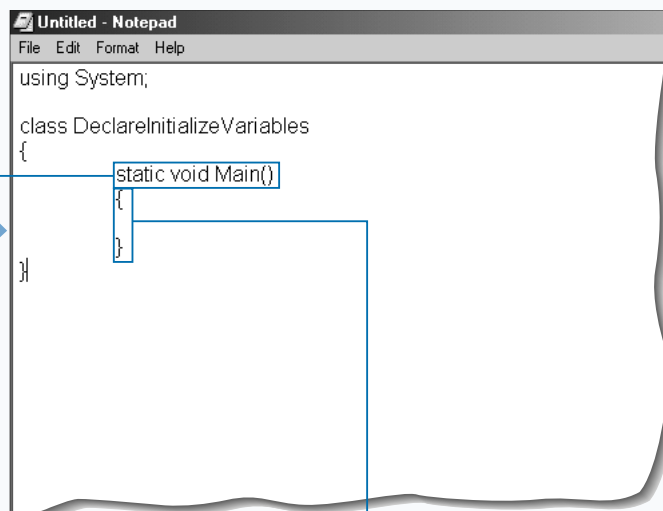
## DECLARE A VARIABLE



```
using System;

class DeclareInitializeVariables
{
}

```



```
using System;

class DeclareInitializeVariables
{
    static void Main()
    {
    }
}

```

- 1 Open your text editor.
- 2 Type **using System;** to import the System namespace and press Enter.

- 3 Type the name of the class you want to create and press Enter.
- 4 Type **{ }**, placing the opening and closing curly braces on separate lines, to specify the body of the class.

- 5 Between the curly braces, type **static void Main()** and press Enter to create the **Main** function.

- 6 Type **{ }**, placing the opening and closing curly braces on separate lines.



## Apply It

You must declare and initialize variables before using them to avoid an error. Failing to initialize a variable produces a runtime error.

### TYPE THIS:

```
namespace DeclareInitializeVariables
{
    using System;

    /// <summary>
    ///     Summary description for ApplyIt.
    /// </summary>
    public class ApplyVariableDeclaration
    {
        public void ApplyIt()
        {
            string sTest;

            // Executing this line will give you the following error:
            Console.WriteLine (sTest);
        }
    }
}
```

### RESULT:

Compile error = "Use of unassigned local variable 'sTest'"

```
using System;

class DeclareInitializeVariables
{
    static void Main()
    {
        string sStatement;
        sStatement = "Make a statement";
    }
}
```

**7** Between the curly braces, type two statements to declare and initialize a variable on separate lines and then press Enter.

```
using System;

class DeclareInitializeVariables
{
    static void Main()
    {
        string sStatement;
        sStatement = "Make a statement";
        int iCount = 0;
    }
}
```

**8** Type a statement to declare and initialize a variable on the same line and press Enter.

# INITIALIZE A VARIABLE

After you declare a variable, you initialize it by giving it a value. This value can come directly from the result of an operation by adding two numbers or concatenating a string, from the return of a method, or from the value of a property or field on an object.

When choosing a name for a variable, you need to give it a meaningful name. This lets you know what is in its storage without having to search through the code to find out what data type was used when it was declared. If you are storing someone's first name in a variable, you should give it a name like `strFirstName`. The convention used in

`strFirstName` is Hungarian notation. Using this naming convention tells us the data type of the variable and the classification of the information (a first name).

.NET has a common set of data types that all .NET-compliant languages use. Having a *Common Type System* (CTS) is one of the foundations of the .NET platform that allows cross-language compatibility. The CTS is a formal specification that details how a type is defined. When you initialize a variable, you need to make sure that you pass data that can be stored in that type; otherwise a runtime error will occur.

## INITIALIZE A VARIABLE

```

Untitled - Notepad
File Edit Format Help
using System;

class DeclareInitializeVariables
{
    static void Main()
    {
        string sStatement;
        sStatement = "Make a statement";

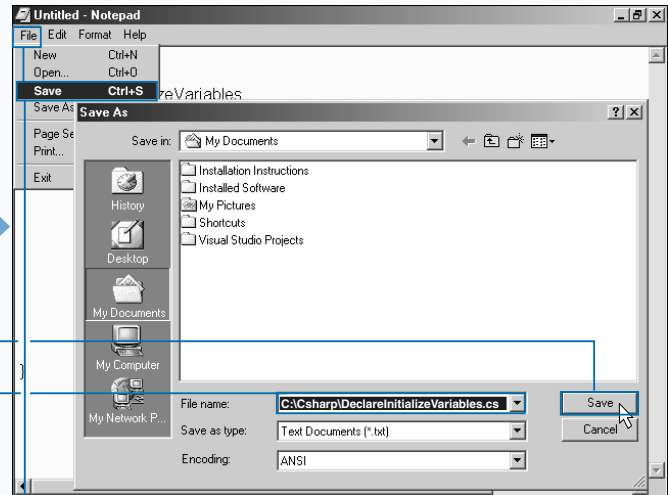
        int iCount = 0;

        int iOddNumbers = 0, iEvenNumbers = 0;

        Console.WriteLine("The value for variables are: sStatement="
            + sStatement + ", iCount =" + iCount);
    }
}
  
```

**1** Type the statement to declare and initialize a multiple variable of the same type on the same line and press Enter.

**2** Add the `Console.WriteLine` function to write the values for the variables to the console screen.



**3** Click File ⇨ Save.

**4** Type a name for the file.

**5** Click Save.

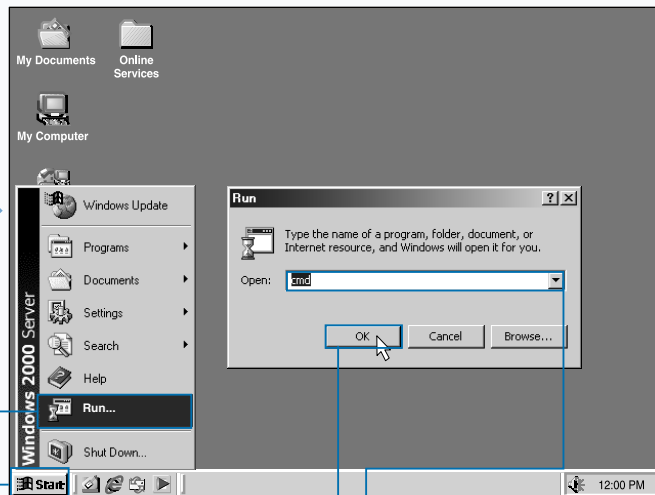
The source file saves to the directory and can now be compiled.

*Note: You can save all of your Console applications in a specific directory (example: C:\CSharp).*

## Extra

You need to know what data types are available to you in C# to properly store data. The following table outlines the intrinsic data types used by C#.

C# DATA TYPE	DESCRIPTION	SAMPLE CODE
Object	The ultimate base type of all other types	<code>object o = null;</code>
String	String type; a string is a sequence of Unicode characters	<code>string s = "hello";</code>
Sbyte (byte)	8-bit signed integral type (unsigned)	<code>sbyte val = 12;</code>
Short (ushort)	16-bit signed integral type (unsigned)	<code>short val = 12;</code>
int (uint)	32-bit signed integral type (unsigned)	<code>int val = 12;</code>
long (ulong)	64-bit signed integral type (unsigned)	<code>long val1 = 12; long val2 = 34L;</code>
float (double)	Single-precision floating point type (double precision)	<code>float val = 1.23F;</code>
bool	Boolean type; either true or false	<code>bool val1 = true; bool val2 = false;</code>
char	Character type; a char value is a Unicode character	<code>char val = 'h';</code>
decimal	Precise decimal type with 28 significant digits	<code>decimal val = 1.23M;</code>

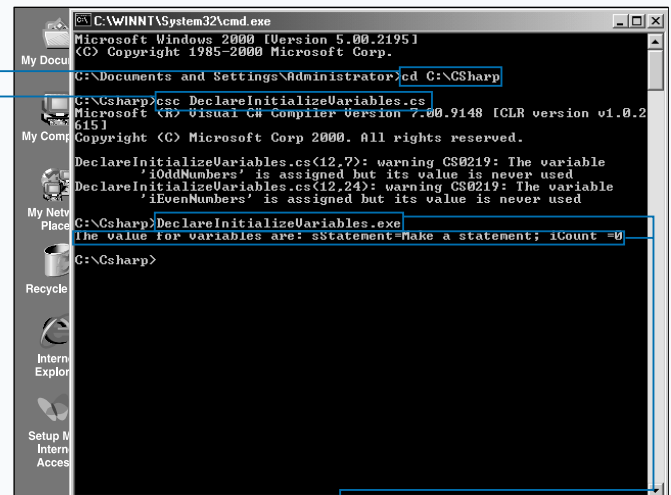


**6** Click Start ⇨ Run to open the Run dialog box.

■ The Run dialog box appears.

**7** Type `cmd` in the Open field.

**8** Click OK.



**9** Change directories to where `DeclareInitializeVariables.cs` is located by using the `cd` command.

**10** Compile the class at the command prompt with the `csc` command.

*Note: See page 36 for more information on compiling and running a file.*

**11** Run the program by typing the name of the executable file and pressing Enter.

■ The program displays the message about the initialized variables.

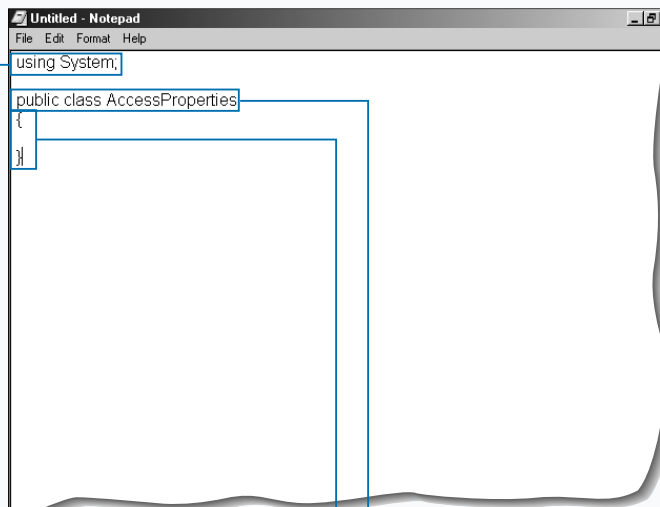
# ACCESS PROPERTIES

You can access the attributes of an object by using properties. A *property* is a member that provides access to an attribute of an object or a class. Examples of properties include the length of a string, the size of a font, the caption of a window, and the name of a customer.

Many objects in the .NET Framework have very useful properties. For example, you can use the `DateTime` object from the `System` class for a few handy properties. You can use the `Now` property to get a date/time stamp for the current date and time, or you can use the `Today` property to get the current date.

Accessing properties requires the class that defines the object to be available for use in your application. When using a .NET implicit object, you will need to make sure that the namespace for that class is imported. Next, you will qualify the class and the property that you want to access. You can do this by fully qualifying the class and its property (for example: `System.DateTime.Now`) or taking a shortcut that does not include the name of the base class that is referenced (for example: `DateTime.Now`).

## ACCESS PROPERTIES



```
using System;

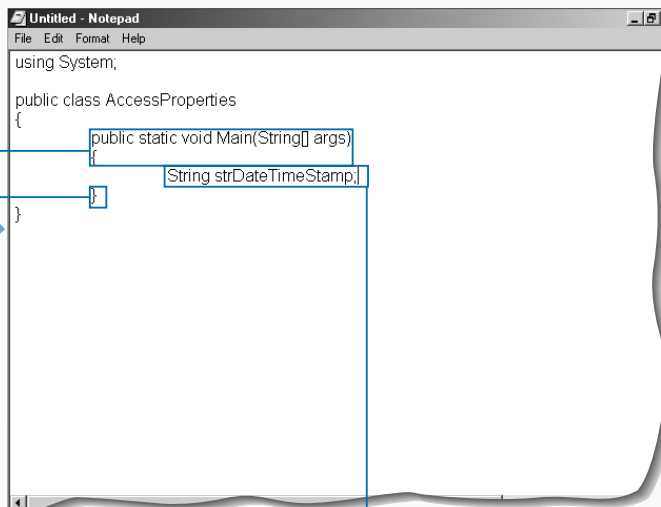
public class AccessProperties
{
}
```

1 Open your text editor.

2 Type **using System;** to import the `System` namespace and press Enter.

3 Type the name of the class you want to create and press Enter.

4 Type `{ }`, placing the opening and closing curly braces on separate lines, to set off the body of the class.



```
using System;

public class AccessProperties
{
    public static void Main(String[] args)
    {
        String strDateTimeStamp;
    }
}
```

5 Between the curly braces, create the `Main` function.

6 Declare variable to hold a `Date Time` string.

**Extra**

Object-oriented languages have three import concepts for working with objects. To work with an object, you will need to know what properties, methods, and events are. This chapter covers these three concepts. Properties describe attributes of your object. For example, you could represent a pencil as an object. Some of the properties of this object would be its length, color, and thickness of lead.

Properties are a natural extension of fields. They are both named members with associated types, and the syntax for accessing fields and properties is the same. In general, you should expose properties instead of public fields. With properties you have better control over the storage and access of the information stored. You will want to use `Get` and `Set` accessors to control reading and writing to properties.

```

AccessProperties.cs - Notepad
File Edit Format Help
using System;

public class AccessProperties
{
    public static void Main(String[] args)
    {
        String strDateTimeStamp;
        strDateTimeStamp = DateTime.Now.ToString();
        Console.WriteLine("The Date and Time is " + strDateTimeStamp);
    }
}

```

```

C:\WINNT\System32\cmd.exe
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\Documents and Settings\Administrator>cd C:\CSharp
C:\CSharp>csc AccessProperties.cs
Microsoft (R) Visual C# Compiler Version 7.00.9148 [CLR version
6151]
Copyright (C) Microsoft Corp 2000. All rights reserved.

C:\CSharp>AccessProperties.exe
The Date and Time is 5/1/2001 4:10:15 PM
C:\CSharp>

```

**7** Access the `Now` property from the `DateTime` class that is part of the "System" Framework class.

**8** Send this value to the console.

**9** Save as the class name.

**10** Compile and run the program.

The program displays the message about the date and time.

*Note: See page 35 for more information on saving a file.*

# MAKE DECISIONS USING CONDITIONAL STATEMENTS

You will typically use conditional statements in your code to enforce business logic. Conditional statements (selection statements in the C# specification) are for making decisions in your code.

You have two main ways to implement conditional code, the `if` and `switch` statements. You will commonly use the `if` statement for a single comparison that has code that needs to execute when the comparison yields `true` and when the comparison yields `false`. The `switch` statement works best when multiple comparisons with one value are used for controlling the execution of code.

Both the `if` and `switch` statements are controlled by Boolean expressions. Boolean expressions yield either

a `true` or `false` value. With the `if` statement, if the Boolean expression evaluates to `true`, the first embedded section of code runs. After this is done, control is transferred to the end of the `if` statement. If the Boolean expression evaluates to `false`, the control then goes to the second embedded section of code. After this second embedded section runs, control goes to the end of the `if` statement.

The `if` and the `switch` statements can both be used to control conditional flow. It is up to you to determine which construct will best solve your programming problem.

## MAKE DECISIONS USING CONDITIONAL STATEMENTS

```
using System;

class MakeStatements
{
}

```

- 1 Open your text editor.
- 2 Type `using System;` to import the System namespace and press Enter.

- 3 Type the name of the class you want to create and press Enter.
- 4 Type `{}`, placing the opening and closing curly braces on separate lines, to set off the body of the class.

```
using System;

class MakeStatements
{
    static void Main(string[] args)
    {
        Console.WriteLine("Enter ONE character to be evaluated: ");
        char cUserinput = (char) Console.Read();
    }
}

```

- 5 Between the curly braces, create the `Main` function that reads in the arguments from the command line.
- 6 Use the `Console.WriteLine` function to write a message to the console for collecting a character.

- 7 Declare a `character` variable and read the user input into the variable.

*Note: Only the first character is read, not the line.*

**Extra**

After evaluating and converting a switch statement to the governing type, you can execute the statement several ways.

IF ...	THEN ...
A constant specified in case a label is equal to the value of the switch expression	Control is transferred to the statement list following the matched case label.
No constant matches the value of the switch expression, and a default label is present	Control is transferred to the statement list following the default label.
No constant matches the value of the switch expression, and no default label is present	Control is transferred to the end point of the switch statement.

**8** Create an `if-then` statement to determine whether the input was a number.

**9** Write the appropriate message to the console.

**10** Save the file as the class name.

**11** Compile and run the program.

The program displays the message about the character the user input.

*Note: See page 35 for more information on saving a file.*

# WORK WITH ARRAYS

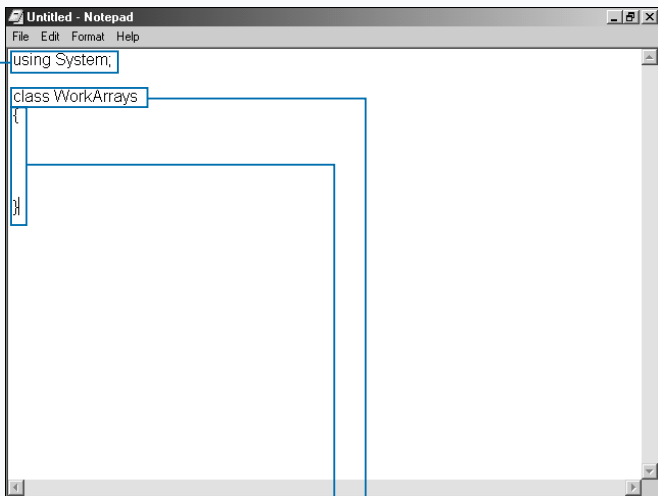
**Y**ou can create arrays when working with a set of variables of the same data type that are related to each other. For example, you may use an array to hold a list of states. Because the state names will all be string data types, you would define a string array of 52 members.

Arrays are a variable type, so you will need to declare and initialize them just like you need to declare and initialize a string variable type. When declaring an array, you will determine the data type needed for storing members of the array and you will determine the number of members in the array.

With arrays, the default lower bound of the array is 0. So when you access the first member of the State List array, `strStateList[ ]`, you would reference this member with `strStateList[0]`. If this State List array was defined to contain 52 members, then the last member would be referenced as `strStateList[51]`.

Arrays allow you to optimize lines of code. You can do so by iterating through all the members of the array with a standard `For` or `For Each` construct. If you structure your code this way, you do not have to add any lines of code if new members are added to the array.

## WORK WITH ARRAYS



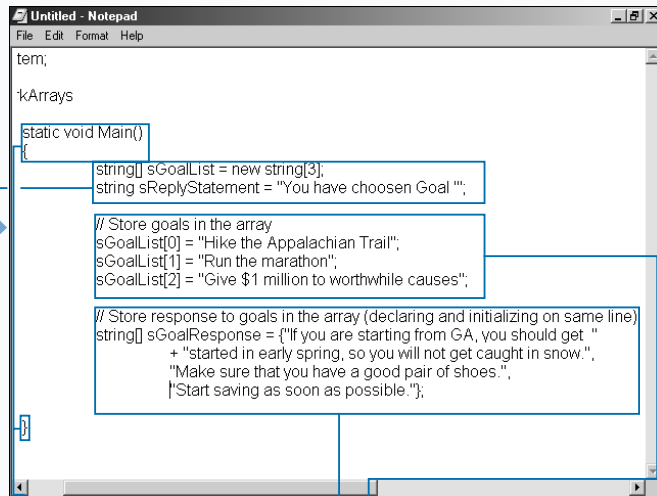
```
using System;

class WorkArrays
{
}

```

**1** Open your text editor.  
**2** Type **using System;** to import the System namespace and press Enter.

**3** Type the name of the class you want to create and press Enter.  
**4** Type `{ }`, placing the opening and closing curly braces on separate lines.



```
term;

kArrays

static void Main()
{
    string[] sGoalList = new string[3];
    string sReplyStatement = "You have chosen Goal ";

    // Store goals in the array
    sGoalList[0] = "Hike the Appalachian Trail";
    sGoalList[1] = "Run the marathon";
    sGoalList[2] = "Give $1 million to worthwhile causes";

    // Store response to goals in the array (declaring and initializing on same line)
    string[] sGoalResponse = {"If you are starting from GA, you should get "
        + "started in early spring, so you will not get caught in snow.",
        "Make sure that you have a good pair of shoes.",
        "Start saving as soon as possible."};
}

```

**5** Between the curly braces, create the **Main** function.  
**6** Declare an **array** variable for holding the goals and a string variable for the question to the user.

**7** Store goals in an array for displaying a message to the console.  
**8** Store responses in another array.



## Apply It

You can use the `Array` class provided in the .NET Framework to manipulate and sort the members of the array.

## TYPE THIS:

```
using System;
namespace ApplyArrays
{
    class ApplyArrays
    {
        static void Main()
        {
            // Initialize an array in same line as declare
            string[] sGoalList = {"Hike the Appalachian Trail",
                "Run a marathon",
                "Give $1 million to worthwhile causes"};
            // Write the members of sGoalList to the console before modifying
            Console.WriteLine("Before sort:\n");
            Console.WriteLine("{0}\n{1}\n{2}\n",sGoalList);
            // Write the sGoalList to the console after sorting
            Array.Sort(sGoalList);
            Console.WriteLine("\nAfter the sort:");
            Console.WriteLine("{0}\n{1}\n{2}\n",sGoalList);
        }
    }
}
```

## RESULT:

Before sort:

Hike the Appalachian Trail

Run a marathon

Give \$1 million to worthwhile causes

After the sort:

Give \$1 million to worthwhile causes

Hike the Appalachian Trail

Run a marathon

```
// Store response to goals in the array (declaring and initializing on s
string[] sGoalResponse = {"If you are starting from GA, you should ge
+ "started in early spring, so you will not get caught in snow.
"Make sure that you have a good pair of shoes.",
"Start saving as soon as possible."};

// Give the user a list of goals to choose from
Console.WriteLine("GOAL LIST");

for(int i = 0; i < sGoalList.Length; i++)
{
    Console.WriteLine("Goal " + i + " - " + sGoalList[i]);
}

// Request the user to choose a goal.
Console.WriteLine(""); // Write an empty line for space
Console.WriteLine("Please choose the number of the "
+ "!goal that you want to achieve [0,1,2]. ");
```

**9** Write the message to the console.

*Note: See page 35 for more information on saving a file.*

**10** Save the file as the class name.

```
C:\WINNT\System32\cmd.exe
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\Documents and Settings\Administrator>cd C:\Csharp
C:\Csharp>dir WorkArrays.cs
Microsoft (R) Visual C# Compiler Version 7.00.9148 [CLR version v1.0.2
615]
Copyright (C) Microsoft Corp 2000. All rights reserved.

WorkArrays.cs(8,10): warning CS8219: The variable 'sReplyStatement' is
assigned but its value is never used

C:\Csharp>WorkArrays.exe
GOAL LIST
Goal 0 - Hike the Appalachian Trail
Goal 1 - Run the marathon
Goal 2 - Give $1 million to worthwhile causes
Please choose the number of the goal that you want to achieve [0,1,2]:
C:\Csharp>
```

**11** Compile and run the program.

The program displays the message about the goal list and options.

# CONTROL LOGIC USING ITERATIVE STATEMENTS

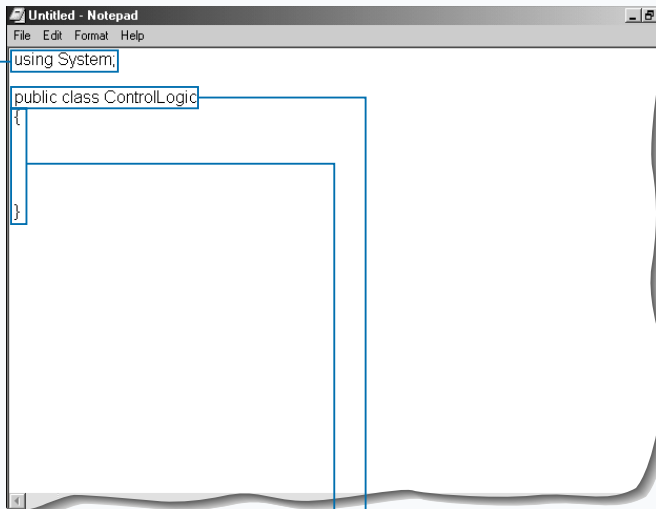
You need to use iterative statements when a section of code needs to execute more than once. For example, after creating an array, you may use an iterative statement to work with every member of that array. There are multiple ways to implement an iterative statement. Your choice is based on the requirements of the logic you are implementing.

There are four iteration statements to choose from: `while`, `do`, `for`, and `foreach`. With the chosen statement, you will need to create a Boolean expression that is evaluated each time the loop is executed. After choosing a statement, you will write code that is embedded within the statement.

Each iterative statement handles a loop differently. The `while` statement will execute 0 or more times. The `do` statement will execute 1 or more times. The `foreach` statement is used for enumerating elements in a collection. The `for` statement has more structure than the `while` and `do` statements. You have three optional parameters used to operate loops, which are an initializer, condition, and iterator.

Within all the iterative statements, the embedded code can use either a `break` or `continue` statement. The `break` statement will transfer the control to the end of the iterative statement and stop the iteration. The `continue` statement will transfer control to the end of the iterative statement and then perform another iteration.

## CONTROL LOGIC USING ITERATIVE STATEMENTS



```
using System;

public class ControlLogic
{
}

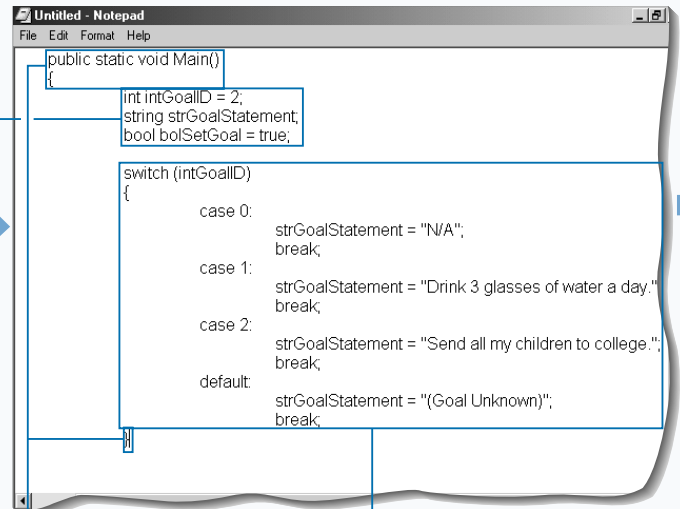
```

1 Open your text editor.

2 Type `using System;` to import the System namespace and press Enter.

3 Type the name of the class you want to create and press Enter.

4 Type `{ }`, placing the opening and closing curly braces on separate lines, to set off the body of the class.



```
public static void Main()
{
    int intGoalID = 2;
    string strGoalStatement;
    bool bolSetGoal = true;

    switch (intGoalID)
    {
        case 0:
            strGoalStatement = "N/A";
            break;
        case 1:
            strGoalStatement = "Drink 3 glasses of water a day.";
            break;
        case 2:
            strGoalStatement = "Send all my children to college.";
            break;
        default:
            strGoalStatement = "(Goal Unknown)";
            break;
    }
}

```

5 Between the curly braces, create the `Main` function.

6 Declare variables for use by the `case` statement and the `if-then` logic.

7 Create a `case` statement.

## Apply It

You will find that `while` statements work well with applying business logic. The following is a simple example of using a `while` loop.

### TYPE THIS:

```
using System;
class ApplyWhileStatements
{
    public static void Main()
    {
        int n = 1;

        while (n < 6)
        {
            Console.WriteLine("Current value of n is {0}", n);
            n++;
        }
    }
}
```

### RESULT:

```
Current value of n is 1
Current value of n is 2
Current value of n is 3
Current value of n is 4
Current value of n is 5
```

```
ControlLogic - Notepad
File Edit Format Help

case 1:
    strGoalStatement = "Drink 3 glasses of water a day.";
    break;
case 2:
    strGoalStatement = "Send all my children to college.";
    break;
default:
    strGoalStatement = "(Goal Unknown)";
    break;
}
if (bolSetGoal)
{
    Console.WriteLine("The goal of '" + strGoalStatement + "' was set.");
}
else
{
    Console.WriteLine("No goal was set.");
}
```

**8** Create an `if-then` statement.

**9** Use the `writeline` method to send the appropriate value to the console.

**10** Save as the class name.

*Note: See page 35 if you need more information on saving a file.*

```
C:\WINNT\System32\cmd.exe
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\Documents and Settings\Administrator>cd C:\CSharp
C:\CSharp>csc ControlLogic.cs
Microsoft (R) Visual C# Compiler Version 7.00.9148 [CLR version
515]
Copyright (C) Microsoft Corp 2000. All rights reserved.

C:\CSharp>ControlLogic.exe
The goal of 'Send all my children to college.' was set.

C:\CSharp>
```

**11** Compile and run the program.

The program displays the message about the goal set.

# CONCATENATE A STRING

You will find many cases that you need to programmatically build a string by concatenating two or more strings together. You can work with two variables of string data type and join them together into a single string. Programmers call this process *string concatenation*. You will use the `+` operator in C# to concatenate the two strings together.

You can format a message to the user by using a combination of string variables and literals that are in quotation marks. For example, say you have a variable like `strUserName` and you have populated that variable with the User's Name. You want to format a

message which welcomes the user using their name. To do this, you would use the statement `"Welcome, " + strUsername`.

There are several ways to concatenate two or more string sources. You can use either the arithmetic operator `+` operator or the `+=` assignment operator. The `+` operator would be used to combine strings in the order that they appear in the expression. The `+=` assignment would be used to append a string to an existing string. Remember that as you append your strings, you will have to include the spacing inside the double quotes of your string to have proper spacing between words.

## CONCATENATE A STRING

```

using System;

public class ConcatenateString
{
}
  
```

```

using System;

public class ConcatenateString
{
    public static void Main()
    {
        string strFirst = "set";
        string strSecond = "goals";
        string strSimpleStatement = strFirst + strSecond;
    }
}
  
```

- 1 Open your text editor.
- 2 Type **using System;** to import the System namespace and press Enter.

- 3 Type the name of the class you want to create and press Enter.
- 4 Type `{ }`, placing the opening and closing curly braces on separate lines, to set off the body of the class.

- 5 Between the curly braces, create the **Main** function.

- 6 Declare two string variables.
- 7 Concatenate the strings together.

## Apply It

You can shorten the code required to concatenate a string to itself by using the += assignment operator. The sample below passes arguments into a console application from the command line and builds a single string that puts all the command line parameters into one string.

### COMPILE THIS:

```
using System;
public class ApplyStringConcatenation
{
    public static void Main(String[] args)
    {
        String strDynamicString = "";

        // Loop through the arguments and concatenate into one string
        for(int i = 0; i < args.Length; i++)
        {
            strDynamicString += args[i] + " ";
        }

        // Write the result of the concatenated string to the console
        Console.WriteLine("Your goal list is: " +
            strDynamicString);
    }
}
```

### TYPE THIS:

```
<program name>.exe "Run a
marathon" "Go to top of
Empire State Building"
```

### RESULT:

Your goal list is: Run a marathon; Go to top of Empire State Building;

```
ConcatenateString
public static void Main()
{
    string strFirst = "set";
    string strSecond = "goals";

    string strSimpleStatement = strFirst + strSecond;

    string strBetterStatement = "You should " + strFirst + " " + strSecond + " monthly";

    Console.WriteLine("A simple sample of concatenating two strings: '" +
        strSimpleStatement + "'");
    Console.WriteLine("A sample created from concatenating variables with text: '" +
        strBetterStatement + "'");
}
```

**8** Concatenate the strings within a statement.

**9** Write the two statements to the console.

**10** Save as the class name.

*Note: See page 35 for more information on saving a file.*

```
C:\WINNT\System32\cmd.exe
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\Documents and Settings\Administrator>cd C:\Csharp

C:\Csharp>csc ConcatenateString.cs
Microsoft (R) Visual C# Compiler Version 7.00.9148 (CLR version v1.0.2
5151
Copyright (C) Microsoft Corp 2000. All rights reserved.

C:\Csharp>ConcatenateString.exe
A simple sample of concatenating two strings: 'setgoals'
A sample created from concatenating variables with text: 'You should s
et goals monthly'

C:\Csharp>
```

**11** Compile and run the program.

The program displays the message about the concatenated strings.

# CONVERT A VARIABLE

You will want to convert variables when performing operations that require all variables to be the same data type. For example, suppose you want to add numbers together, and the numbers are held in string variables. Some languages, like Visual Basic 6, will do an implicit conversion, but not in the case of C#. Another example of where you may want to convert is using a method on a class. For example, the `Response.Write` method expects a string data type to be passed for the first parameter. If another data type is passed, a runtime error will occur.

Many functions are available to convert from one data type to another data type. So, the first thing to do

when converting from one data type to another data type is to look up the appropriate function for the conversion. An example of a function that you will use often is the function that converts to a string. To use this method, you simply use the `ToString()` method call at the end of the variable.

There are two different types of conversions: an implicit conversion and an explicit conversion. An implicit conversion is done when you cast another variable into a variable of a different data type. Be careful when you do this, because sometimes the variable you are casting into cannot hold the original variable.

## CONVERT A VARIABLE

```
using System;

class ConvertVariables
{
}
```

```
using System;

class ConvertVariables
{
    static void Main()
    {
        byte bteValue = 3;
        int intValue = bteValue;

        Console.WriteLine("The value for the byte is {0} and the value for the integer is {1}", bteValue, intValue);
    }
}
```

1 Open your text editor.

2 Type **using System;** to import the System namespace and press Enter.

3 Type the name of the class you want to create and press Enter.

4 Type `{ }`, placing the opening and closing curly braces on separate lines, to set off the body of the class.

5 Between the curly braces, create the `Main` function.

6 Declare a `byte` variable and initialize the value to 3.

7 Declare an `integer` variable and initialize the value to the `byte` variable created.

8 Output the two values to the console.

*Note: This is an example of an implicit conversion.*

## Apply It

The following example represents a class definition that does several different types of conversions. The first conversion changes the data type from a number to a string. The second conversion demonstrates a conversion from a string to a number. Finally, the dates are converted into different formats.

### Example:

```
using System;
public class ApplyConversion
{
    static void Main()
    {
        string strExample = "1.2";
        string strDate;

        // This is how to convert a string to number
        double dblValue = Double.Parse(strExample);

        // This is how numerics are converted to strings
        string strDoubleValue = dblValue.ToString();

        // This is conversion of full dates to other data types
        strDate = DateTime.Now.ToLocalTime().ToString();
        strDate = DateTime.Now.ToLongTimeString();
        strDate = DateTime.Now.ToShortDateString();
    }
}
```

```
using System;

class ConvertVariables
{
    static void Main()
    {
        byte bteValue = 3;
        int intValue = bteValue;

        Console.WriteLine("The value for the byte is {0} and the value for
        the integer is {1}", bteValue, intValue);

        long lngMaxValue = Int64.MaxValue;
        int intOverflowValue = (int) lngMaxValue;

        Console.WriteLine("Converting a maximum long, {0}, to an integer
        is {1}", lngMaxValue, intOverflowValue);
    }
}
```

**9** Declare a **long** variable and initialize the value to the maximum value for a long variable.

**10** Declare an **integer** variable and initialize the value to the byte variable created.

*Note: This is an example of an explicit conversion.*

**11** Output the two values to the console.

**12** Save as the class name.

*Note: See page 35 for more information on saving a file.*

```
Microsoft Windows [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\Documents and Settings\Administrator>cd C:\CSharp

C:\CSharp>csc ConvertVariables.cs
Microsoft (R) Visual C# Compiler Version 7.00.9148 (CLR version v1.0.2
615)
Copyright (C) Microsoft Corp 2000. All rights reserved.

C:\CSharp>ConvertVariables.exe
The value for the byte is 3 and the value for the integer is 3
Converting a maximum long, 9223372036854775807, to an integer is -1

C:\CSharp>
```

**13** Compile and run the program.

The program displays the message about the **byte** and **long** variable conversions.

# ENUMERATE A COLLECTION

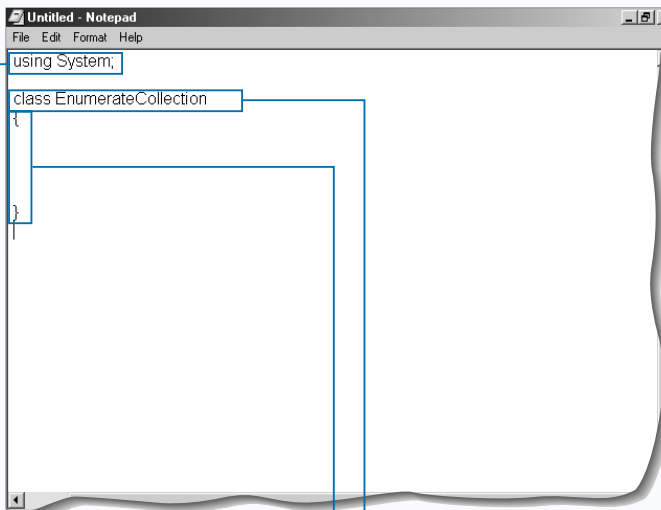
**C**ollections give you a sophisticated way to work with a group of items. One collection type is a single-dimension array. When programming with a single-dimension array, you can use the `foreach` statement to iterate through the collection.

Looping through a collection, you can use any of the iteration statements that are available in C#. If you use a `while` or a `do` statement, you will have to manually move through the collection using the `MoveNext` method and then check to see when you have reached the end of the collection. This will work, but you will find that using the `foreach` statement will eliminate the work required to move to the next member and to keep track of where you are in the collection.

The `foreach` statement automatically increments the position in the collection and automatically stops after the last member has been evaluated. The `foreach` statement will also put you at the first member of that collection when entering the `foreach` statement the first time.

Also, with the collection, the statement includes an iteration variable that holds an instance of the member at the current position. You are most likely to use this instance in your embedded statement.

## ENUMERATE A COLLECTION

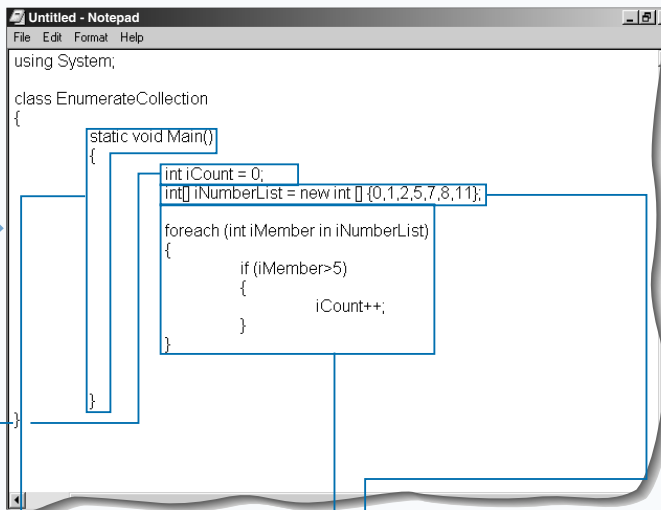


```
using System;

class EnumerateCollection
{
}
```

- 1 Open your text editor.
- 2 Type **using System;** to import the System namespace and press Enter.

- 3 Type the name of the class you want to create and press Enter.
- 4 Type `{ }`, placing the opening and closing curly braces on separate lines, to set off the body of the class.



```
using System;

class EnumerateCollection
{
    static void Main()
    {
        int iCount = 0;
        int[] iNumberList = new int [] {0,1,2,5,7,8,11};

        foreach (int iMember in iNumberList)
        {
            if (iMember>5)
            {
                iCount++;
            }
        }
    }
}
```

- 5 Between the curly braces, create the `Main` function.
- 6 Declare an `integer` variable and initialize the value to 0.
- 7 Declare a collection of integers.
- 8 Add the `foreach` statement to display all the members over 5.



## Apply It

You can use a `for` statement instead of a `foreach` statement. Note how much extra programming you must do to make this work.

### Example:

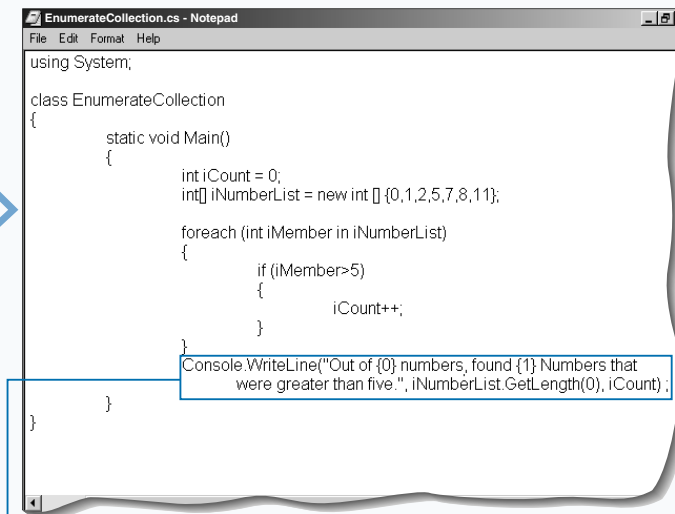
```
static void Main()
{

    int iCount = 0;
    int[] iNumberList = new int [] {0,1,2,5,7,8,11};

    for (int i=0; i<iNumberList.Length; i++)

    {
        if (iNumberList[i]>5)
        {
            iCount++;
        }
    }

    Console.WriteLine("Out of {0} numbers, found {1} Numbers that were greater
than five.",
        iNumberList.GetLength(0), iCount) ;
}
```



```
EnumerateCollection.cs - Notepad
File Edit Format Help
using System;

class EnumerateCollection
{
    static void Main()
    {
        int iCount = 0;
        int[] iNumberList = new int [] {0,1,2,5,7,8,11};

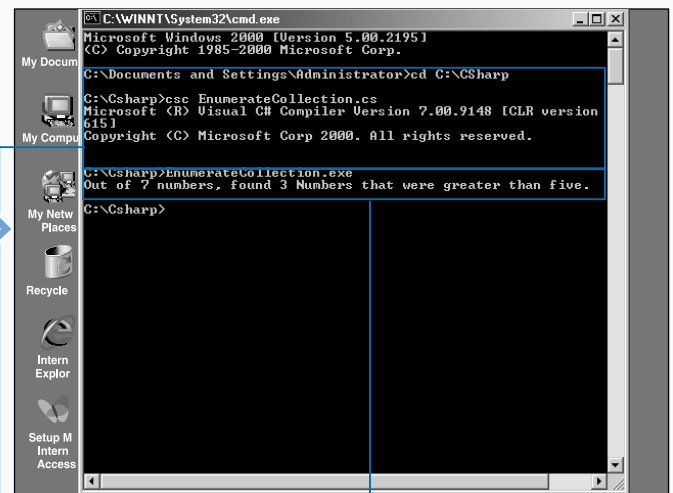
        foreach (int iMember in iNumberList)
        {
            if (iMember>5)
            {
                iCount++;
            }
        }

        Console.WriteLine("Out of {0} numbers, found {1} Numbers that
were greater than five.", iNumberList.GetLength(0), iCount) ;
    }
}
```

**9** Output the results to the console.

**10** Save as the class name.

*Note: See page 35 for more information on saving a file.*



```
C:\WINNT\System32\cmd.exe
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\Documents and Settings\Administrator>cd C:\CSharp

C:\CSharp>cmd /c csc EnumerateCollection.cs
Microsoft (R) Visual C# Compiler Version 7.00.9148 [CLR version
615]
Copyright (C) Microsoft Corp 2000. All rights reserved.

C:\CSharp>EnumerateCollection.exe
Out of 7 numbers, found 3 Numbers that were greater than five.

C:\CSharp>
```

**11** Compile and run the program.

The program displays the message about the results.

# DECLARE AND USE METHODS

**M**ethods are members of an object or class that implement a computation or action. You can use methods to hold a section of code that may be used more than one time. Methods are built into classes using method declarations.

Many objects in the .NET Framework classes contain useful methods. For example, the .NET Framework `DateTime` class has several methods that can be used, such as the `ToLongDateString` method, which converts a `Date` or `DateTime` to a long version of a `Date` (for example: Thursday, May 30, 2002).

Programmers also use methods to encapsulate functionality. For example, you could represent a

printer as an object. One of the methods could be `PrintPage`. This method could take in a parameter that is a stream of data to be printed and could return a value that indicates whether the data printed successfully. Consider a `Calculator` object as another example. This object could have several methods like `Add`, `Subtract`, `Multiply`, and `Divide`. The `Subtract` method could take in two parameters that are integer data types and return an integer. The `Subtract` method would contain functionality that would take one parameter and subtract the other parameter and return the result.

## DECLARE AND USE METHODS

```
using System;

class DeclareUseMethods
{
}

```

- 1 Open your text editor.
- 2 Type **using System;** to import the System namespace and press Enter.

- 3 Type the name of the class you want to create and press Enter.
- 4 Type `{}`, placing the opening and closing curly braces on separate lines, to set off the body of the class.

```
using System;

class DeclareUseMethods
{
    public static void Main()
    {
        string sCurrentDate;
        char cUserInput;

        Console.WriteLine("CHOOSE A FORMAT FOR THE CURRENT DATE:");
        Console.WriteLine("0 - Short Date");
        Console.WriteLine("1 - Short Time");
        Console.WriteLine("2 - Day of the Week");
        Console.WriteLine("3 - Date and Time");
    }
}

```

- 5 Between the curly braces, create the `Main` function.
- 6 Declare a `string` variable to hold the date, and a `character` variable to hold the user input.
- 7 Write a message about the types of formats to the console.

## Apply It

You have many methods to leverage from objects in the .NET Framework.

### TYPE THIS:

```
using System;
public class ApplyNETFramework
{
    public static void Main()
    {
        Console.WriteLine(System.DateTime.Now.ToLongTimeString());
    }
}
```

### RESULT:

output would be: 10:12:12 PM

```
using System;

class DeclareUseMethods
{
    public static void Main()
    {
        string sCurrentDate;
        char cUserInput;

        Console.WriteLine("CHOOSE A FORMAT FOR THE CURRENT DATE:");
        Console.WriteLine("0 - Short Date");
        Console.WriteLine("1 - Short Time");
        Console.WriteLine("2 - Day of the Week");
        Console.WriteLine("3 - Date and Time");

        cUserInput = (char) Console.Read();
        sCurrentDate = GetFormattedDate(cUserInput);
    }
}
```

**8** Read the input from the console.

**9** Call the `GetFormattedDate` function with the user input and put into the variable for the current date.

```
class DeclareUseMethods
{
    public static void Main()
    {
        string sCurrentDate;
        char cUserInput;

        Console.WriteLine("CHOOSE A FORMAT FOR THE CURRENT DATE:");
        Console.WriteLine("0 - Short Date");
        Console.WriteLine("1 - Short Time");
        Console.WriteLine("2 - Day of the Week");
        Console.WriteLine("3 - Date and Time");

        cUserInput = (char) Console.Read();
        sCurrentDate = GetFormattedDate(cUserInput);

        Console.WriteLine("The current date on the system in short date format is " + sCurrentDate + "");
    }
}
```

**10** Write the message about the current date to the console.

CONTINUED

# DECLARE AND USE METHODS

Working in an object-oriented language, you will find three important concepts: properties, methods, and events. *Methods* hold operations that can have 0 to many input parameters and 0 to one return values. Methods enable the developer to hide (encapsulate) the difficulty of a coding task by placing complicated sections of code in a method. If the signature of the method (input parameters and return values) is well thought out, the developer can change how he implements the code and not affect the consumer of that method. If the developer has to change the signature of that method, the consumer of the method will not function any more, although this

problem can be solved via overloaded functions. A signature change includes any of the following: adding or subtracting input parameters or return values; changing the data type of an input parameter or return value.

When you are new to ASP.NET development, you use many methods that are available from the .NET Framework. As you get comfortable with developing in ASP.NET, you will find that you will start creating your own custom methods. Taking this approach will reduce the number of lines you will have to write in your code and make the code itself easier to maintain.

## DECLARE AND USE METHODS (CONTINUED)

```

Untitled - Notepad
File Edit Format Help

string sCurrentDate;
char cUserInput;

Console.WriteLine("CHOOSE A FORMAT FOR THE CURRENT DATE");
Console.WriteLine("0 - Short Date");
Console.WriteLine("1 - Short Time");
Console.WriteLine("2 - Day of the Week");
Console.WriteLine("3 - Date and Time");

cUserInput = (char) Console.Read();

sCurrentDate = GetFormattedDate(cUserInput);

Console.WriteLine("The current date on the system in short date format is: "
    + sCurrentDate + ".");
}

public static String GetFormattedDate(char cChoice)
{
    string sReturn;
}

```

**11** Create the `GetFormattedDate` function, which returns a string.

```

Untitled - Notepad
File Edit Format Help

Console.WriteLine("3 - Date and Time");
cUserInput = (char) Console.Read();

sCurrentDate = GetFormattedDate(cUserInput);

Console.WriteLine("The current date on the system in short date format is: "
    + sCurrentDate + ".");
}

public static String GetFormattedDate(char cChoice)
{
    string sReturn;

    switch (cChoice.ToString())
    {
    }

    return sReturn;
}
}

```

**12** Declare a `string` variable to hold the return value.

**13** Create a `case` statement to handle the different types of formats.

**Extra**

A method has four possible signatures.

**Example:**

```
using System;

public class DeclareUseMethodsExtra
{
    // Public operation that takes no parameters and has no return
    public static void MySubRoutine()
    {
        // code goes here
    }

    // Public operation that takes parameters and has no return
    public static void MySubRoutine(int intParam)
    {
        // code goes here
    }

    // Public operation that takes no parameters and has a return
    public static string MySimpleFunction()
    {
        // code goes here
        return "MyReturn";
    }

    // Public operation that takes in parameters and has a return
    public static int MyAddFunction(int intParam1, int intParam2)
    {
        return intParam1 + intParam2;
    }
}
```

```
switch (cChoice.ToString())
{
    case "0":
        sReturn = DateTime.Now.ToShortDateString();
        break;
    case "1":
        sReturn = DateTime.Now.ToShortTimeString();
        break;
    case "2":
        sReturn = DateTime.Now.DayOfWeek.ToString();
        break;
    case "3":
        sReturn = DateTime.Now.ToString();
        break;
    default:
        sReturn = "{not supported}";
        break;
}
```

**14** Create the different cases for the user input, formatting the return variable appropriately.

**15** Save as the class name.

*Note: See the section "Write Your First C# Application," earlier in this chapter, if you need more information on saving a file.*

```
C:\WINNT\System32\cmd.exe
Microsoft Windows [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\Documents and Settings\Administrator>cd C:\CSharp
C:\CSharp>exe DeclareUseMethods.cs
Microsoft (R) Visual C# Compiler Version 7.00.9148 [CLR version 615]
Copyright (C) Microsoft Corp 2000. All rights reserved.

C:\CSharp>DeclareUseMethods.exe
CHOOSE A FORMAT FOR THE CURRENT DATE:
0 - Short Date
1 - Short Time
2 - Day of the Week
3 - Date and Time
The current date on the system in short date format is '5/2/2001'

C:\CSharp>DeclareUseMethods.exe
CHOOSE A FORMAT FOR THE CURRENT DATE:
0 - Short Date
1 - Short Time
2 - Day of the Week
3 - Date and Time
The current date on the system in short date format is '9:10 AM'

C:\CSharp>DeclareUseMethods.exe
CHOOSE A FORMAT FOR THE CURRENT DATE:
0 - Short Date
1 - Short Time
2 - Day of the Week
3 - Date and Time
```

**16** Compile and run the program.

The program displays the options for different dates and times and displays results for the current date and time.

# IMPLEMENT EXCEPTION HANDLING

When programmatically working with errors, you need to understand exception handling. No matter how good a developer you are, you cannot avoid runtime errors. For example, suppose your program tries to read a file that does not exist. How would you handle this? You would first try to access the file and then, if there is a failure, you would want to have code that runs if the error occurs. If the error occurred, you may want to ask the user to pick a new path for that file and then try again.

If you do not code for handling errors, your code would either stop executing or would move on to the next executable statement. Moving on to the next statement is acceptable in some cases, but not all.

The exception handling in C# is performed by using the keywords `try` and `catch`. The code that you want to “try” goes in a block of code after the `try` statement. After that try code block, you would put in a `catch` statement. The code that you want to execute in the event that an error occurs would go into a block of code that is after the `catch` statement.

## IMPLEMENT EXCEPTION HANDLING

```
using System;

class ImplementExceptionHandling
{
}

```

```
using System;

class ImplementExceptionHandling
{
    static void Main(string[] args)
    {
        ImplementExceptionHandling x = new
            ImplementExceptionHandling();

        try
        {
            string s = null;
            x.EvaluateString(s);
        }

        catch (ArgumentNullException e)
        {
            Console.WriteLine("'0' was the exception.",
                e.Message);
        }
    }
}

```

- 1 Open your text editor.
- 2 Type **using System;** to import the System namespace and press Enter.

- 3 Type the name of the class you want to create and press Enter.
- 4 Type {}, placing the opening and closing curly braces on separate lines, to set off the body of the class.

- 5 Between the curly braces, create the **Main** function.
- 6 Create a new variable of type **ImplementExceptionHandling**.

- 7 Use the **try** statement to create a new **string** variable set to **null**, and use this variable to call **EvaluateString**.
- 8 Use the **catch** statement to write out the exception.

## Extra

Exceptions can be thrown in two different ways. A throw statement can be executed programmatically or it can happen at runtime where the processing of the C# statement causes the error (like dividing by zero). Below is a table of common exception classes:

EXCEPTION	DESCRIPTION
System.OutOfMemoryException	Thrown when an attempt to allocate memory (via new) fails.
System.StackOverflowException	Thrown when the execution stack is exhausted by having too many pending method calls; typically indicative of very deep or unbounded recursion.
System.NullReferenceException	Thrown when a null reference is used in a way that causes the referenced object to be required.
System.InvalidCastException	Thrown when an explicit conversion from a base type or interface to a derived type fails at runtime.
System.ArrayTypeMismatchException	Thrown when a store into an array fails because the actual type of the stored element is incompatible with the actual type of the array.
System.IndexOutOfRangeException	Thrown when an attempt to index an array via an index that is less than zero or outside the bounds of the array fails.
System.ArithmeticException	A base class for exceptions that occur during arithmetic operations, such as DivideByZeroException and OverflowException.

```

DeclareUseMethods.cs - Notepad
File Edit Format Help
}
catch (ArgumentNullException e)
{
    Console.WriteLine("'0' was the exception.", e.Message);
}
}

public string EvaluateString(string s)
{
    if (s == null)
    {
        throw new ArgumentNullException();
    }
    else
    {
        return s;
    }
}
}

```

**9** Create a function called `EvaluateString`.

**10** Use this function to raise an `ArgumentNullException` error.

**11** Save as the class name.

*Note: See page 35 for more information on saving a file.*

```

C:\WINNT\System32\cmd.exe
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\Documents and Settings\Administrator>cd C:\CSharp
C:\Csharp>csc ImplementExceptionHandling.cs
Microsoft (R) Visual C# Compiler Version 7.00.9148 [CLR version
615]
Copyright (C) Microsoft Corp 2000. All rights reserved.

C:\Csharp>ImplementExceptionHandling.exe
'Value cannot be null.' was the exception.

C:\Csharp>

```

**12** Compile and run the program.

The program displays the error that was raised.

# CONVERT A CONSOLE APPLICATION TO AN ASP.NET WEB PAGE

You can migrate code from a console application to an ASP.NET Web Page. The coding is very similar, but you need to adjust code that deals with the user interface. With the console application, your user interface is the command line. With the ASP.NET Web page, your user interface is a Web browser.

You can reuse many parts of the console application. In fact, except for the code pertaining to the user interface, much of the code will remain the same. The process for converting the console application is to create a user interface in HTML that gathers

appropriate information. For example, you will use a drop-down list box in this task to prompt the user for the type of format in which they would like to see the date.

Console applications are closer to a procedural style of programming. The user interface is very simple, and users do not have many ways to interact with the program, except for command-line parameters. Moving from console applications to ASP.NET applications, you will need to understand how to use event handlers. To learn more about event handlers, see page 32.

## CONVERT A CONSOLE APPLICATION TO AN ASP.NET WEB PAGE

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void SubmitBtn_Click(object Source, EventArgs e) {
    spanMessage.InnerHtml=GetFormattedDate(selectSuggestions.Value);
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3>Convert to Web Page</H3>
<FORM ID="formConvert" RUNAT="Server">
<P/>
<SELECT ID="selectSuggestions" RUNAT="Server">
  <OPTION VALUE="0">Short Date</OPTION>
  <OPTION VALUE="1">Short Time</OPTION>
  <OPTION VALUE="2">Day of the Week</OPTION>
  <OPTION VALUE="3">Date and Time</OPTION>
</SELECT>
<INPUT TYPE="Submit" VALUE="Submit" OnServerClick="SubmitBtn_Click"
RUNAT="Server">
  
```

**1** Open the `ConvertToWebPageTemplate.aspx` template file from the CD-ROM.

```

Untitled - Notepad
File Edit Format Help
    }
    + sCurrentDate + " ";
}

public static String GetFormattedDate(char cChoice)
{
    string sReturn;
    switch (cChoice.ToString())
    {
        case "0":
            sReturn = DateTime.Now.ToShortDateString();
            break;

        case "1":
            sReturn =
            DateTime.Now.ToShortTimeString();
            break;

        case "2":
            sReturn =
            DateTime.Now.DayOfWeek.ToString();
            break;

        case "3":
  
```

**2** Open the console application source file, `DeclareUseMethods.cs`, from the CD-ROM.

**3** Copy the `GetFormattedDate` function from `DeclareUseMethods.cs` source file.



**Extra**

Leverage the use of components when you write a console application that eventually becomes an ASP.NET application. When implementing your application functionality, encapsulate logical units of code behind methods in the classes that make up the component. If you program in this way, you will be able to reuse the code when migrating to a new user interface (Windows application or ASP.NET application). Also, if you put your code into components, you will be able to share these components with future or existing applications.

You will typically not choose to write console applications if you build a production application. There are cases in which you would choose a console application. Some examples of useful console-type applications include testing out programming syntax, building quick test harnesses, building administrative applications, or building debugging tools.

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
public static String GetFormattedDate(char cChoice){
    string sReturn;
    switch (cChoice.ToString()){
        case "0":
            sReturn = DateTime.Now.ToShortDateString();
            break;

        case "1":
            sReturn = DateTime.Now.ToShortTimeString();
            break;

        case "2":
            sReturn = DateTime.Now.DayOfWeek.ToString();
            break;

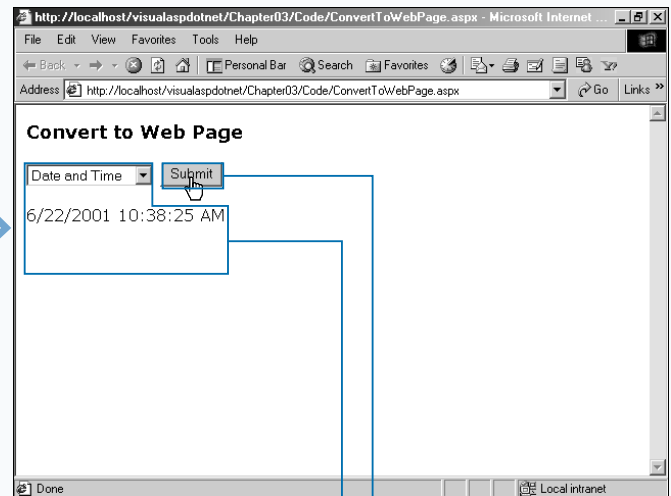
        case "3":
            sReturn = DateTime.Now.ToString();
            break;

        default:
    
```

**4** Paste the `GetFormattedDate` function into the `<SCRIPT>` section of the ASP.NET Web page.

■ Scroll down the page to view the `SubmitBtn_Click` function in the page, which calls the `GetFormattedDate` function.

■ A `Submit` button calls the `SubmitBtn_Click` function.



**5** Save the file and request from the Web server.

**6** Select a date format.

**7** Click the `Submit` button.

■ The date appears in the format selected.

# INTRODUCTION TO HTML CONTROLS

You can take standard HTML elements and control these elements on your Web server with HTML server controls. This gives you control

over attributes of these elements while processing server-side code.

## BASICS OF HTML SERVER CONTROLS

*HTML server controls* are essentially HTML elements that the server can process. This processing can occur before sending the Web page to the user and/or when the page is posted back to the server. All HTML server controls (also known as *HTML controls*) map directly to an HTML element. Also, “the properties of” almost every HTML control are identical to the corresponding HTML element’s attribute.

HTML controls are defined in the `System.Web.UI.HtmlControls` namespace. You can create an HTML control in most cases simply by adding the `RUNAT="Server"` attribute within the tag of the HTML element. If you incorrectly set the `RUNAT` attribute on the HTML element, you lose all server-side processing capabilities. You want to give each HTML control a unique ID attribute so you have a way to reference the control in your server-side code.

You can set attributes for the HTML controls to establish control properties and to handle events. Property attributes configure how the control appears and behaves as an HTML element on a Web page. In most cases, the property attributes map directly to the standard attributes on the HTML 4.0 element. For example, the `ID` attribute on an HTML control renders in the client’s browser as an `ID` attribute on the HTML element, too. As for handling events, you distinguish events as attributes on an HTML control to map an HTML control’s event to a procedure that will process the event. For example, when working with the `HTMLButton` control, you can add an attribute

`onServerClick` to map to a function that is called when the user clicks the button.

HTML server controls derived from the `HtmlInput` abstract class need to be within an `HtmlForm` control. For all HTML server controls, overlapping tags are not allowed — you need to make sure that the HTML tags are properly closed and cleanly nested.

The .NET Framework provides classes to handle the most commonly used HTML elements. For HTML elements that corresponding HTML controls, lacking the `HtmlGenericControl` class. You can programmatically read and write to attributes on the HTML element and map events to server-side code. Note that any attribute declared on an HTML server control is added to the `HtmlGenericControl`’s `Attribute` collection and can be manipulated in server-side code. For example, with a `<body ID="Body" RUNAT="Server">` element, you can programmatically attribute on a body tag change the `bgcolor`. `netframeworkoffers` with the following code:

```
Body.Attributes["bgcolor"] = "blue";)
```

All HTML controls are derived from the `HtmlControl` abstract class. Within this abstract class, there are two other abstract classes, `HtmlContainerControl` and `HtmlInputControl`, which contain all HTML controls, except for the `HtmlImage` class, which is derived directly from the `HtmlControl` abstract class.

## CLASSIFICATIONS OF HTML CONTROLS

**All HTML Controls**

On the `HtmlControl` abstract class, you can find the following properties, which are commonly used across all HTML controls.

**Attributes (Read)** — Collection of all attribute name and value pairs expressed on a server control tag within a selected ASP.NET page

**Disabled (Read/Write)** — A value that indicates whether the disabled attribute is included when the browser renders an HTML control. Including this attribute makes the control read-only.

**Style (Read)** — All Cascading Style Sheet (CSS) properties applied to a specified HTML server control in an `.aspx` file

**TagName (Read)** — The element name of a tag containing a `RUNAT="Server"` attribute

**HTML Input Controls**

`HtmlInputControl` derived controls map to the standard HTML input elements that are part of an HTML form. These HTML elements function without a separate closing tag. They include a `Type` attribute that defines the type of input control they render on a Web page.

`HtmlInputControl` derived classes share the following properties:

**Name (Read/Write)** — A unique identifier name for the `HtmlInputControl`

**Value (Read/Write)** — The contents of an input control

**Type (Read)** — The type of `HtmlInputControl`

**HTML Container Controls**

`HtmlContainerControl`'s derived controls map to HTML elements requiring opening and closing tags, such as the `<select>`, `<a>`, `<button>`, and `<form>` elements.

`HtmlContainerControl`'s derived classes share the following common properties:

**InnerHtml (Read/Write)** — The HTML content found between the opening and closing tags of the specified HTML control

**InnerText (Read/Write)** — All text between the opening and closing tags of the specified HTML control

**HTML Image Control**

The `HtmlImage` server control is the only concrete class derived directly from the `HtmlControl` abstract class. The common properties for this control are:

**Align (Read/Write)** — The alignment of the image relative to other Web page elements

**Alt (Read/Write)** — The alternative caption that the browser displays if an image is unavailable or currently downloading to the user's browser

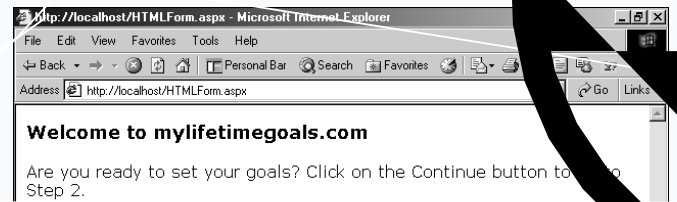
**Border (Read/Write)** — The width of a frame for an image

**Height (Read/Write)** — The height of the image

**Src (Read/Write)** — The source of the image file to display

**Width (Read/Write)** — The width of the image





# CREATE A FORM BUTTON

You can use a form button to control actions that take place on an HTML form, to submit a form to the Web server, or to reset the contents of the form. You can provide the capability to submit a form by placing a submit button (`<input TYPE="submit">`) on the form. To enable the user to reset all controls on a form, you can use the reset button (`<input TYPE="reset">`).

If you want to process a server-side `HTMLForm`, you typically use the `HTMLInput` control. To ensure that the `HTMLInput` control sends the form to the server, set the `TYPE` attribute to `Submit`. You can write code

that will run only when the page is submitted to the server by implementing the `OnServerClick` event.

You can create a simple login page to test for a static password by adding a couple of `HTMLInputText` controls on the page. See page 74 for more details. You can create two types of text boxes: one for text and another for a password. After creating these, you can place a `<Submit>` button and a `<Reset>` button on the form. You implement both of these buttons using the `HTMLInputButton` control.

## CREATE A FORM BUTTON

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Please login to the secured area. You can use "goals" as a guest password.

<FORM RUNAT="Server">
</FORM>

</FONT>
</BODY>
</HTML>

```

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Please login to the secured area. You can use "goals" as a guest password.

<FORM RUNAT="Server">
Enter Name: <INPUT ID="inputName" TYPE="Text" SIZE="40" RUNAT="Server">
</P>
Enter Password: <INPUT ID="inputPassword" TYPE="Password" SIZE="40"
RUNAT="Server">
<P>
<INPUT TYPE="Submit" VALUE="Submit" OnServerClick="SubmitBtn_Click"
RUNAT="Server">
</P>
<SPAN ID="spanMessage" style="color:red" RUNAT="Server"></SPAN>
</FORM>

</FONT>

```

**1** Open the `GenericTemplate.aspx` from the Code Templates directory.

**2** Add a heading for the page.

**3** Add a message to the user.

**4** Add a form to the page with a `RUNAT` attribute set to `"Server"`.

**5** Add an `HTMLInputText` control for the name and the password and label them.

**6** Add an `HTMLInputButton` on the page and have it call the `SubmitBtn_Click` function for the `OnServerClick` event.

**7** Add an `HTMLSpan` control on the page to display the results of the login.

**Apply It**

For the convenience of your user, you can add a clear button to the page that will clear out the contents of the form. There is an input of type-Reset that resets the form to original values. This example clears all contents despite the original value.

**TYPE THIS:**

```
void ResetBtn_Click(object Source, EventArgs e) {
    inputName.Value = "";
    inputPassword.Value = "";
}
```

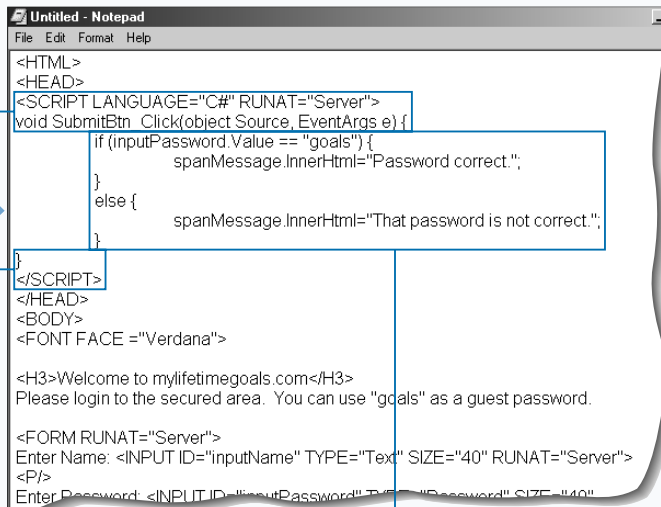
Then, add an `HTMLInputButton` with the `TYPE` attribute set to `Button`. For this control, use the `OnServerClick` event to call the `ClearBtn_Click` event.

```
<INPUT TYPE="Button" OnServerClick="ClearBtn_Click" RUNAT="Server">
```



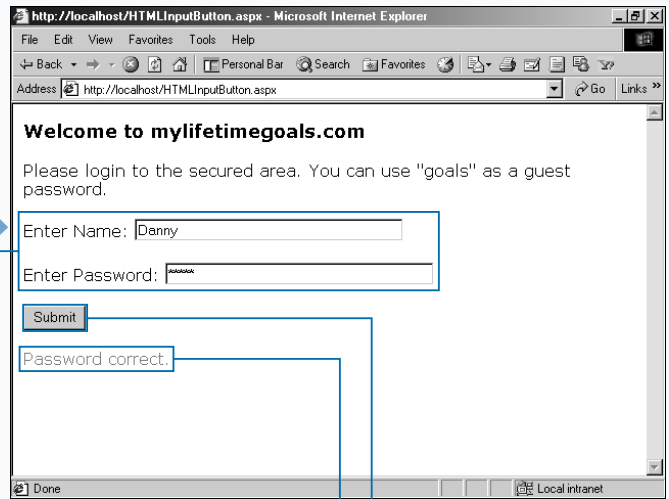
**RESULT:**

A Web page that allows the user to click the Clear button to clear the form



**8** Create a `SubmitBtn_Click` function to check the password.

**9** Use an `if` statement to check the password and display the message using the `HTMLSpan` control on the page.



**10** Save the file and request it from the Web server.

*Note: See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.*

**12** Click the Submit button.

A message appears notifying you if your password is correct.

*Note: Try entering an incorrect password.*

**11** Type in a name and a correct password.

# CREATE AN HTML 4.0 BUTTON

If you are working with a user that has a Web browser that supports the HTML 4.0 `<button>` element, you can use the `HTMLButton` control to create a button with some nice features. Note that the `<button>` element is defined in the HTML 4.0 specification; therefore, it is supported only in Microsoft Internet Explorer version 4.0 and above (Other popular browsers like Navigator and Opera, currently do not support the `<button>` element). With HTML 4.0, you can use some client-side events to customize the look, feel, and behavior of buttons.

An `HTMLButton` control needs to reside on an `HTMLForm` control within your page to fully utilize this

control. As when adding other controls, give each of the `HTMLButton` controls an ID and set the `RUNAT` attribute equal to `Server`.

You can use the two DHTML events, `onMouseOver` and `onMouseOut`, to set button properties. For example, you can change a button's background color whenever a user positions the mouse cursor over the button. This feature lets the user know that he or she can click the button. You can use the `onMouseOver` event to change the background color. To change the background color back to the original setting when the user moves the mouse cursor away from the button, use the `onMouseOut` event.

## CREATE AN HTML 4.0 BUTTON

```

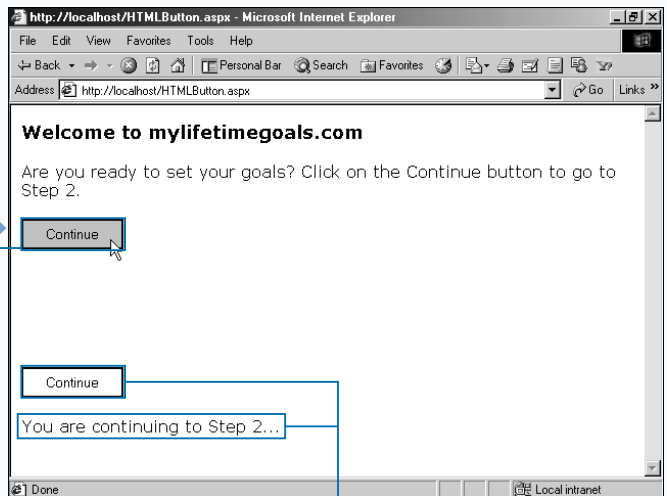
Untitled - Notepad
File Edit Format Help

<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
    void Button_OnClick(object Source, EventArgs e) {
        spanMessage.InnerHtml="You are continuing to Step 2...";
    }
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Are you ready to set your goals? Click on the Continue button to go to Step 2.

<FORM RUNAT="Server">
<P>
<BUTTON ID="buttonContinue" RUNAT="Server" onServerClick="Button_OnClick"

STYLE="background-color:#FFFFFF;border-color:black;height=30;width:100"
onMouseOver="this.style.backgroundColor=#C0C0C0"
onMouseOut="this.style.backgroundColor=#FFFFFF">
Continue
</BUTTON>
</P>
  
```



- 1 Open the `WelcomeTemplate.aspx` template from the Code Templates directory.
- 2 Add a Continue button with the `HTMLButton` control to the form and call the `Button_OnClick` function for the `onServerClick` event.

- 3 Set the `STYLE` attribute for the button.
- 4 Set the background color of the button for the `onMouseOver` event.
- 5 Set the background color for the button for the `onMouseOut` event.

- 6 Save the file and request it from the Web server.

*Note: See pages 20 to 25 for instructions on saving a file to the Web server and requesting the file using the IIS Admin.*

- 7 Position your mouse over the button.

- The button background color changes.
- 8 Click Continue and move the mouse off the button.
- The button background color changes back.
- A continue message appears.



# CREATE A GRAPHICAL BUTTON

**Y**ou can use an image as a form-submitting button as well. If you target HTML 4.0 compatible Web browsers, you can also employ the `onMouseOver` and `onMouseOut` events to change the image dynamically.

The `HTMLInputImage` must be on an `HTMLForm` control, on an ASP.NET page. As with adding other Web controls, you must specify the `ID` attribute and set the `RUNAT` attribute equal to `Server`. You may also want to set the `<SRC>` attribute for the

`HTMLInputImage` control. This setting tells the Web browser where to locate the image. For the `SRC` attribute, specify the path to the image relative to the root directory of your Web server.

You can use the `onMouseOver` and the `onMouseOut` events to set the properties of your images. Using these events, you can change the `SRC` of the image to a new value. When the user moves the mouse cursor off the image, you can change the image back to the original setting by using the `onMouseOut` event.

## CREATE A GRAPHICAL BUTTON

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Button_OnClick(object Source, ImageClickEventArgs e) {
    spanMessage.InnerHtml="You are continuing to Step 2...";
}
</SCRIPT>
</HEAD>
<BODY>
<FONT face="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
Are you ready to set your goals? Click on the Stoplight to go to Step 2.
<FORM RUNAT="Server">
<P>
<INPUT TYPE="Image" ID="inputImage" RUNAT="Server"
OnServerClick="Button_OnClick"
SRC="images/redlight.gif"
onMouseOver="this.src='images/greenlight.gif'"
onMouseOut="this.src='images/redlight.gif'"
>
</P>
  
```

**1** Open the `WelcomeTemplate.aspx` template from the Code Templates directory.

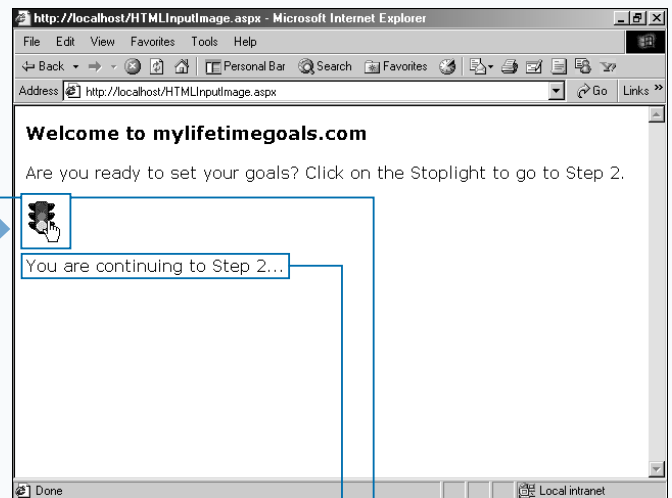
**2** Add an `HTMLInputImage` control to the form and call the `Button_OnClick` function.

**3** Set the `SRC` attribute for the button.

**4** Set the image for the `onMouseOver` event.

**5** Set the image for the `onMouseOut` event.

**6** Change the parameters on the `Button_OnClick` function from `EventArgs` to `ImageClickEventArgs`.



**7** Save the file and request it from the Web server.

*Note:* See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.

■ The light is red.

**8** Move your mouse over the button and click.

■ The light changes to green and a continue message appears.

# REQUEST SINGLE LINE INPUT

A text box is a common control that is used on forms for enabling users to enter text into a form. You can use an ASP.NET page to process your text boxes as server controls, enabling you to work with the text box's properties, such as the length and type, in your code.

To indicate that the text boxes are `HtmlInputText` controls rather than ordinary text boxes, add an `ID` attribute and a `Runat` attribute set to `Server` and make sure that it is on an `HtmlForm` control. You should specify what type of text box it is. Two valid text box types are `Password` and `Text`.

You can create a simple login page to test for a static password. To do this, you can add a couple of `HtmlInputText` controls to the page. The first control lets the user enter a name and has the `Type` property set to `Text` to indicate that it is an ordinary text box. The second control has the `Type` set to `Password` to specify that it is input for a password. For security purposes the password text box displays asterisks, rather than the characters you are typing.

## REQUEST SINGLE LINE INPUT

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Please login to the secured area. You can use "goals" as a guest password.

<FORM RUNAT="Server">
</FORM>

</FONT>
</BODY>
</HTML>

```

**1** Open the `GenericTemplate.aspx` template from the Code Templates directory.

**2** Add a heading for the page.

**3** Add a message to the user.

**4** Add an `HtmlForm` control to the page.

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Please login to the secured area. You can use "goals" as a guest password.

<FORM RUNAT="Server">
Enter Name: <INPUT ID="inputName" TYPE="Text" SIZE="40" RUNAT="Server">
<P/>
Enter Password: <INPUT ID="inputPassword" TYPE="Password" SIZE="40"
RUNAT="Server">
<P/>
<INPUT TYPE="Submit" VALUE="Submit" OnServerClick="SubmitBtn_Click"
RUNAT="Server">
<P/>
<SPAN ID="spanMessage" style="color:red" RUNAT="Server"></SPAN>
</FORM>

</FONT>

```

**5** Add `HtmlInputText` controls for the name and the password and label them.

**6** Add an `HtmlInputButton` on the page and have it call the `SubmitBtn_Click` function for the `OnServerClick` event.

**7** Add an `HtmlSpan` control on the page to display the results of the login.

**Extra**

You can create a read-only text box by setting the **Disabled** property of the **HTMLInputText** control to **true**.

**Example:**

```
Enter Name: <INPUT ID="inputName" TYPE="Text"
SIZE="40" RUNAT="Server" VALUE="Guest"
DISABLED="true">
```

You can programmatically hide the text boxes from the user by setting the **Visible** property of the **HTMLInputText** control equal to **false**.

**Example:**

```
Enter Name: <INPUT ID="inputName"
TYPE="Text" SIZE="40" RUNAT="Server"
VISIBLE="false">
```

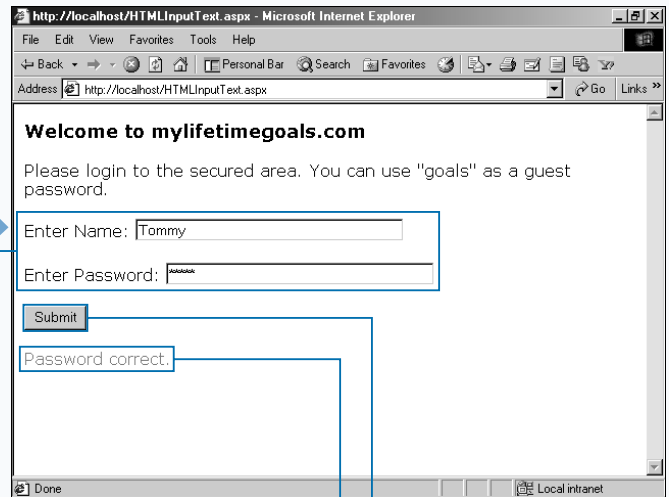
You can limit the number of characters that a user can type into a text box. The **MAXLENGTH** property is used for this purpose.

**Example:**

```
Enter Name: <INPUT ID="inputName" TYPE="Text"
SIZE="40" RUNAT="Server" MAXLENGTH="40">
```

```
Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void SubmitBtn_Click(object Source, EventArgs e) {
    if (inputPassword.Value == "goals") {
        spanMessage.InnerHtml="Password correct.";
    }
    else {
        spanMessage.InnerHtml="That password is not correct.";
    }
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
Please login to the secured area. You can use "goals" as a guest password.
<FORM RUNAT="Server">
Enter Name: <INPUT ID="inputName" TYPE="Text" SIZE="40" RUNAT="Server">
<P/>
Enter Password: <INPUT ID="inputPassword" TYPE="Text" SIZE="40" RUNAT="Server"
MAXLENGTH="40" VALUE="" />

</FORM>
</BODY>
</HTML>
```



**8** Create a function called **SubmitBtn\_Click** to check the password.

**9** Add an **if** statement to check the password and display the message using the **HTMLSpan** control on the page.

**10** Save the file and request it from the Web server.

*Note: See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.*

**12** Click the Submit button.

A message appears notifying you if your password is correct.

**11** Type in a name and the incorrect password.

# REQUEST MULTIPLE LINE INPUT

You can use the `<textarea>` tag to create input boxes that have more than one row. This gives the user more space to type input on the form. You can implement this as an `HTMLTextArea` control so that you can easily access the contents of the control, as well as set the properties of the control.

An `HTMLTextArea` control needs to reside on an `HTMLForm` control within your page to take full advantage of this control. As with adding other controls, you should give each of the `HTMLTextArea` controls an ID and set the `RUNAT` attribute equal to `Server`.

You can specify the number of columns and the number of rows for the control to size it properly. Use `ROWS` and `COLS` properties to modify the height and width respectively. You can modify the text of the `HTMLTextArea` by using the `InnerHTML` or `InnerText` properties. Use `InnerHTML` if you desire to format the text when you update the `HTMLTextArea`.

You can gather suggestions from users by giving them an `HTMLTextArea` control to fill in on a server form. You should also add an `HTMLInputButton` and call a function using that control's `onServerClick` event. The suggestions can be echoed back to the user by setting the `InnerHTML` property of the `<span>` tag.

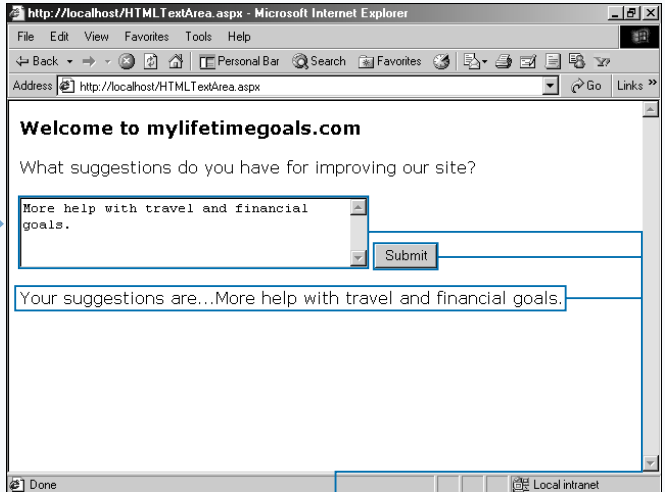
## REQUEST MULTIPLE LINE INPUT

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
    void SubmitBtn_Click(object Source, EventArgs e) {
        spanMessage.InnerHtml="Your suggestions are..." +
        textareaSuggestions.Value;
    }
</SCRIPT>
</HEAD>
<BODY>
<FONT face="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
What suggestions do you have for improving our site?

<FORM RUNAT="Server">
<P/>
<TEXTAREA ID="textareaSuggestions" COLS="40" ROWS="4" RUNAT="Server" />
<INPUT TYPE="Submit" VALUE="Submit" OnServerClick="SubmitBtn_Click"
RUNAT="Server">
<P/>
<SPAN ID="spanMessage" RUNAT="Server" />
</FORM>
  
```



**1** Open the `SuggestionsTemplate.aspx` template from the Code Templates directory.

**2** Add an `HTMLTextArea` control to the form and set the `COLS` and `ROWS` attributes.

**3** Add the code in the `SubmitBtn_Click` function to echo back the contents of the `HTMLTextArea` control.

**4** Save the file and request it from the Web server.

*Note: See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.*

**5** Fill in a suggestion.

**6** Click the Submit button.

A message appears displaying the suggestion.

# REQUEST BOOLEAN INPUT

Check boxes are convenient when you need to have the user respond to a yes/no or true/false question. The `HTMLInputCheckBox` control gives you a server-side control to work with when you need to ask these types of questions. The `HTMLInputCheckBox` control is similar to the `HTMLInputRadioButton` control, but is used more often when you want the user to select zero to many options from a list of options. For example, you could have a registration form, and on that form you could ask for one or more of the person's hobbies by providing check boxes. You should use the `HTMLInputRadioButton` control when you want the user to select only one option from a list of options.

You can initialize the check box to be either checked or not checked. When the form is submitted, you can see if the control was checked by looking at the `Checked` property of the control. If the `Checked` property is `true`, the user has checked the control.

You can set the `VALUE` attribute of the `HTMLInputCheckBox` to either be a key or the actual value of what the check box is representing. For example, setting the `VALUE` attribute as `Have more goals to choose from` displays this in the message that is echoed back to the user.

## REQUEST BOOLEAN INPUT

```

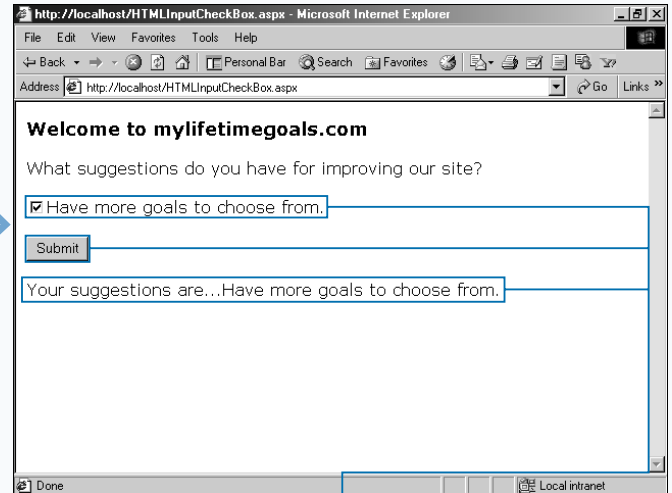
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void SubmitBtn_Click(object Source, EventArgs e) {
    if (inputSuggestions.Checked == true) {
        spanMessage.InnerHtml="Your suggestions are..." +
inputSuggestions.Value;
    }
    else {
        spanMessage.InnerHtml = "You have no suggestions.";
    }
}
</SCRIPT>
</HEAD>
<BODY>
<FONT face="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
What suggestions do you have for improving our site?

<FORM RUNAT="Server">
<P>
<INPUT ID="inputSuggestions" TYPE="Checkbox" RUNAT="Server" VALUE="Have
more goals to choose from."/>Have more goals to choose from.
  
```

- 1 Open the `SuggestionsTemplate.aspx` template from the Code Templates directory.
- 2 Add an `HTMLInputCheckBox` control to the form and set the `VALUE` attribute.

- 3 Add the code in the `SubmitBtn_Click` function to send an appropriate message back to the user.



- 4 Save the file and request it from the Web server.

The ASP.NET Web page appears.

- 5 Check the check box.
  - 6 Click the Submit button.
- A message appears displaying the suggestion.

*Note:* See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.

# REQUEST A SELECTION FROM A GROUP

Sometimes you may want to have a user select a single option from a group of choices. In these cases, you can use the `<input type="radio">` tag to create a radio button. The HTML control in ASP.NET that represents this HTML is the `HTMLInputRadioButton` control. By using the HTML control, you can now work with it on the server.

After you create your `HTMLForm`, you can add a series of multiple `HTMLInputRadioButton` controls to your `HTMLForm`. Each control must have a unique value for the `ID` attribute, but the `NAME` attribute for a group of controls should be the same. By doing this,

the user can select only one option from the group. When you want to inspect information on a unique member of the radio button group, you can access it by the ordinal reference in the array created (`radio[0].checked` checks the first member in the radio button array).

You can set the radio button that is initially selected by setting the `CHECKED` attribute to `true`. If this is not set, none of the radio buttons are selected until the user clicks a radio button. You can ask questions which have multiple choices for an answer and force the user into selecting one of the options by using the `HTMLInputRadioButton` control.

## REQUEST A SELECTION FROM A GROUP

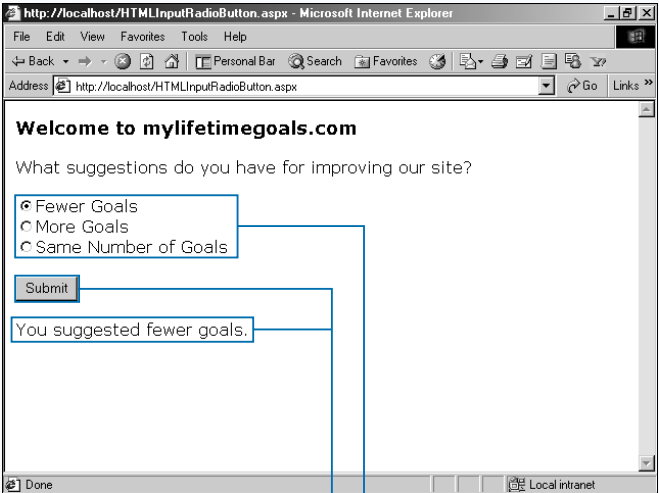
```

<SCRIPT LANGUAGE="C#" RUNAT="Server">
void SubmitBtn_Click(object Source, EventArgs e) {
    if (radioSuggestionsLess.Checked == true)
        spanMessage.InnerHtml = "You suggested fewer goals.";
    else if (radioSuggestionsMore.Checked == true)
        spanMessage.InnerHtml = "You suggested more goals.";
    else if (radioSuggestionsSame.Checked == true)
        spanMessage.InnerHtml = "You suggested same number of goals.";
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
What suggestions do you have for improving our site?

<FORM RUNAT="Server"><P/>
<INPUT TYPE="Radio" ID="radioSuggestionsLess" NAME="Suggestions"
RUNAT="Server"/>Fewer Goals<BR>
<INPUT TYPE="Radio" ID="radioSuggestionsMore" NAME="Suggestions"
RUNAT="Server"/>More Goals<BR>
<INPUT TYPE="Radio" ID="radioSuggestionsSame" NAME="Suggestions"
RUNAT="Server"/>Same Number of Goals<BR>

```



**1** Open the `SuggestionsTemplate.aspx` template from the Code Templates directory.

**2** Add multiple `HTMLInputRadioButton` controls to the form and set `NAME` attributes.

**3** Add the code in the `SubmitBtn_Click` function to send an appropriate message back to the user.

**4** Save the file and request it from the Web server.

The ASP.NET Web page appears.

**5** Check the first radio button.

**6** Click the Submit button.

A message appears displaying the suggestion.

*Note: See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.*

# REQUEST INPUT FROM A DROP-DOWN LIST

Another option for allowing a user to select a single option from a group of options is using the `<select>` tag. This tag creates a drop-down list box from which the user can select a single value. You can use the `HTMLSelect` control to implement the `<select>` tag. The `<select>` tag is also useful when you have a large number of options and little space to display the options.

To fully leverage this control, you could use the data-binding capabilities. For example, if you have an array of values, you can bind to this array by using the

`Datasource` property and the `DataBind` method of the `HTMLSelect` control.

When creating the `HTMLSelect` control programmatically, you can use the `SELECTEDINDEX` attribute to specify which option the user sees when the list box appears. If this is not set, the first option displays. If you want to enable users to select more than that one item from the list, you can include the controls `MULTIPLE` attribute and set it equal to `true`.

Like the `HTMLInputRadioButton` control, you can ask questions which have multiple options for answers and allow only one of the options to be chosen.

## REQUEST INPUT FROM A DROP-DOWN LIST

```

<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
    void SubmitBtn_Click(object Source, EventArgs e) {
        spanMessage.InnerHtml="Your suggestion is..." + selectSuggestions.Value +
    }
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimgoals.com</H3>
What suggestions do you have for improving our site?

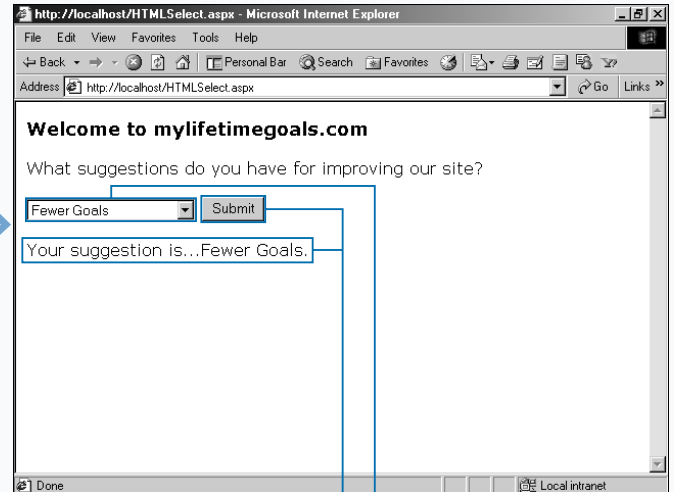
<FORM RUNAT="Server">
<P>
<SELECT ID="selectSuggestions" RUNAT="Server">
    <OPTION>Fewer Goals</OPTION>
    <OPTION>More Goals</OPTION>
    <OPTION>Same Number of Goals</OPTION>
</SELECT>
<INPUT TYPE="Submit" VALUE="Submit" OnServerClick="SubmitBtn_Click"
RUNAT="Server">

```

**1** Open the `SuggestionsTemplate.aspx` template from the Code Templates directory.

**2** Add an `HTMLSelect` control and a number of options to the form.

**3** Add the code in the `SubmitBtn_Click` function to send an appropriate message back to the user.



**4** Save the file and request it from the Web server.

The ASP.NET Web page appears.

**5** Click  and select the first option from the drop-down list.

**6** Click the Submit button.

A message appears displaying the suggestion.

*Note:* See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.



# CREATE A LINK

You can work with the anchor tag, `<a>`, programmatically by using the `HTMLAnchor` control. This control gives you the ability to both set and get information from the `<a>` tags that you create on your ASP.NET pages. When using the `<a>` tag as an HTML server control, it must have an opening and closing tag.

The `HREF` attribute for the anchor tag is the URL that the user gets sent to when clicking the anchor tag. The anchor tag surrounds the text, or HTML, that will be hyperlinked in the user's Web browser. To build your anchor tags dynamically, you can use the combination of a data repeater and data binding to

create parts of the anchor tag. To properly configure a bound anchor tag, you should bind the `HREF` attribute and the text between the `<a>` and `</a>` tags.

While the page is loading, you can set the properties for an `HTMLAnchor` control. For example, you can set the `HREF` property to equal the URL that you want the user to navigate upon clicking the link. You can also set the `InnerText` property of the `HTMLAnchor` control. This text is what the user clicks to request another resource on the Web server.

## CREATE A LINK

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Be sure to visit the Home Page

</FONT>
</BODY>
</HTML>

```

**1** Open the `GenericTemplate.aspx` template from the Code Templates directory.

**2** Add a heading for the page.

**3** Add a message to the user.

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Be sure to visit the Home Page

<FORM RUNAT="Server">
<P>
<A ID="aWebsite" RUNAT="Server"></A>
</FORM>

</FONT>
</BODY>
</HTML>

```

**4** Add a form to the page with a `RUNAT` attribute set to "Server".

**5** Place an anchor with a `RUNAT` attribute set to "Server" on the page for the user to click.



**Extra**

You can override the redirect of the **HTMLAnchor** control by implementing the **ServerClick** event.

**Example:**

```
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void aWebsite_OnClick(object Source,
EventArgs e) {
aWebsite.InnerText="You clicked?";
}
</SCRIPT>
<A ID="aWebsite" RUNAT="Server"
HREF="http://www.mylifetimegoals.com"
onServerClick="aWebsite_OnClick">
Go To Home Page for My Lifetime Goals Website
</A>
```

You can set the attributes for the anchor tag in the HTML as well.

**Example:**

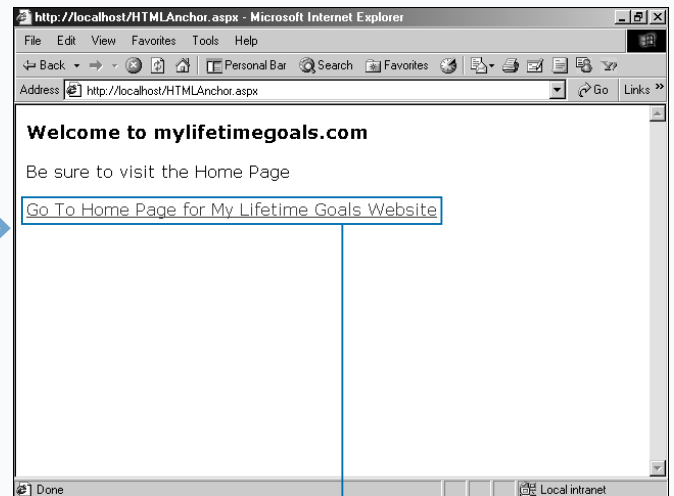
```
<A ID="aWebsite" RUNAT="Server"
HREF="http://www.mylifetimegoals.com">
Go To Home Page for My Lifetime Goals Website
</A>
```

```
Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Page_Load(Object sender, EventArgs e) {
aWebsite.HRef = "http://www.mylifetimegoals.com";
aWebsite.InnerText = "Go To Home Page for My Lifetime Goals Website";
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
Be sure to visit the Home Page
<FORM RUNAT="Server">
<P/>
<A ID="aWebsite" RUNAT="Server"></A>
</FORM>
</FONT>
</BODY>
</HTML>
```

**6** Add the **Page\_Load** function between the **<HEAD>** tags.

**7** Set the **HREF** property of the anchor on the form.

**8** Set the **InnerText** property of the anchor on the form.



**9** Save the file and request it from the Web server.

■ The ASP.NET Web page appears.

*Note: See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.*

■ The anchor appears properly.

# RENDER AN IMAGE

You can use the `HTMLImage` control to have programmatic control over the images on your Web pages. You can use the control to change the properties of the image. For example, you can dynamically change the source of the image so that a graphic can be replaced with another graphic.

You can use the `HTMLImage` control on your ASP.NET Web pages by adding the `ID` and by setting the `RUNAT` attribute to `Server` on the `<img>` tag. You need to set the `SRC` attribute that tells the Web browser the location of the image to load. You could do this in the `Page_Load` event by setting the property

programmatically, or you could set the property by adding the `SRC` attribute in the `<img>` tag.

There are a number of other attributes that you can set for the `HTMLImage` control such as the image border size, the width and height of the image, and the alignment of the image. For example, you can have a list of graphics in a drop-down list box. When the user selects a graphic by name and clicks the `Apply` button, the browser updates the graphic on the page. The initial image that appears is set in the `<img>` tag when designing the page.

## RENDER AN IMAGE

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimgoals.com</H3>
Select your goal type:

</FONT>
</BODY>
</HTML>
  
```

```

Untitled - Notepad
File Edit Format Help
<HEAD>

</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimgoals.com</H3>
Select your goal type:

<FORM RUNAT="Server">
  <SELECT ID="selectGoalImages" RUNAT="Server">
    <OPTION VALUE="travel.gif">Travel</OPTION>
    <OPTION VALUE="career.gif">Career</OPTION>
    <OPTION VALUE="educational.gif">Educational</OPTION>
  </SELECT>
  <INPUT TYPE="Submit" RUNAT="Server" Value="Apply"
  OnServerClick="SubmitButton_Click">
</FORM>

<IMG ID="imageGoalImages"
SRC="/visualaspdotnet/Chapter04/Code/images/travel.gif" RUNAT="Server"/>
</FORM>

</FONT>
  
```

**1** Open the `GenericTemplate.aspx` template from the Code Templates directory.

**2** Add a heading for the page.

**3** Add a message to the user.

**4** Add a form to the page with a `RUNAT` attribute set to "Server".

**5** Add an `HTMLSelect` control on the page so the user can select a goal type.

**6** Add a button on the page for the user to click and use the `OnServerClick` event to call the `SubmitButton_Click` function.

**7** Add an `Image` control on the page and set the source for the image control.

**Apply It**

You can use the ALT property to display a message when the user places the mouse pointer over the image.

```
<FORM RUNAT="Server">
<IMG ID="imageGoalImages"
SRC="/visualaspdotnet/Chapter04/Code/images/travel.gif"
ALT="Picture of a road and a distant planet earth" RUNAT="Server"/>
</FORM>
```

To change the size of the image, you can set the WIDTH and HEIGHT properties of the image.

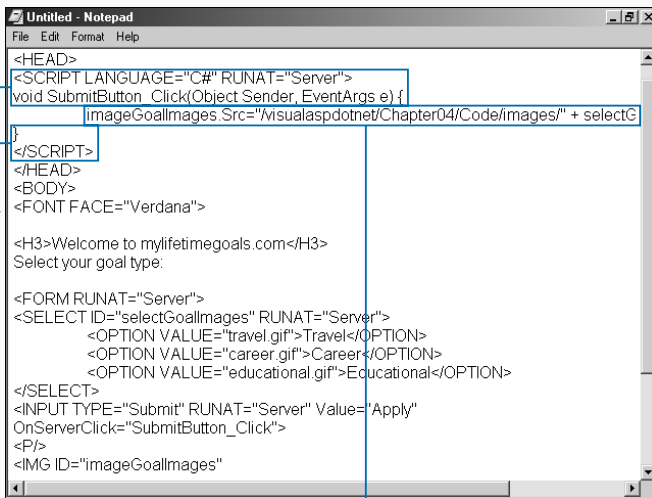
**Example:**

```
<FORM RUNAT="Server">
<IMG ID="imageGoalImages"
SRC="/visualaspdotnet/Chapter04/Code/images/travel.gif"
WIDTH="100" HEIGHT="100" RUNAT="Server"/>
</FORM>
```

You can use the BORDER property to display a border around the image by setting it to a value greater than zero. The default setting of zero displays no border around the image.

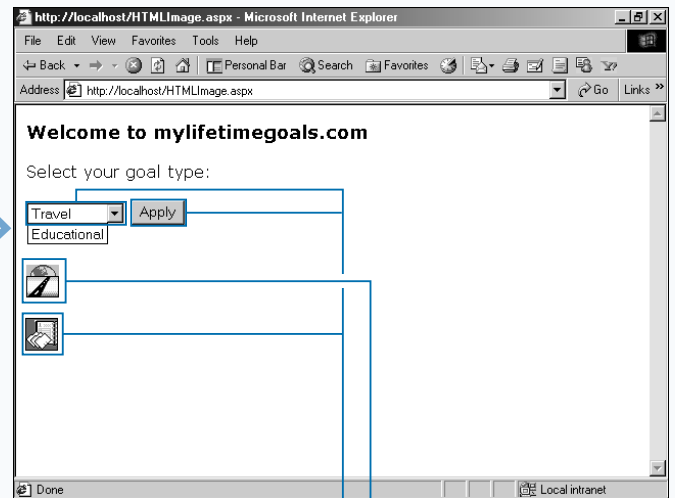
**Example:**

```
<FORM RUNAT="Server">
<IMG ID="imageGoalImages"
SRC="/visualaspdotnet/Chapter04/Code/images/travel.gif"
BORDER="2" RUNAT="Server" />
</FORM>
```



**8** Add the SubmitButton\_Click function.

**9** Set the InnerHTML property of the span on the form in the Sub.



**10** Save the file and request it from the Web server.

The ASP.NET Web page appears.

The Travel graphic appears.

**11** Click [v] and select Educational as the goal type.

**12** Click the Apply button.

The Educational graphic appears.

*Note: See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.*

# BUILD A TABLE

You can use the `HTMLTable` control to dynamically create and modify tables in your ASP pages. You can also use the `HTMLTable` control to set properties for the table, including the background color, cell spacing and padding, and border size and color.

You can create the table in one of the events like the `page_load` event, before the page is generated like the `Page_Load` event. You can place a `<table>` tag on your page like you would do with an ordinary HTML or ASP page. To create an `HTMLTable` control on your ASP.NET page, you need to add the `ID` and `RUNAT` attributes to the table. As with other HTML controls, you need to set the `RUNAT` attribute to `Server`.

## BUILD A TABLE

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Here is your goal list...

<P/>
<FORM RUNAT="Server">
<TABLE ID="tableExample" Border="1" RUNAT="Server" />
</FORM>

</FONT>
</BODY>
</HTML>

```

- 1 Open the `GenericTemplate.aspx` template from the Code Templates directory.
- 2 Add a heading for the page.
- 3 Add a message to the user.

- 4 Add a form to the page.
- 5 Add a table to the page and set the `BORDER` attribute equal to 1.

```

Untitled - Notepad
File Edit Format Help
void Page_Load(Object sender, EventArgs e) {
    int rowcount = 0;
    int numRows = 3;
    int numcells = 1;
    string[] sGoalList = new string[3];
    sGoalList[0] = "Hike the Appalachian Trail";
    sGoalList[1] = "Run a marathon";
    sGoalList[2] = "Give $1 million to worthwhile causes";
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Here is your goal list...

<P/>
<FORM RUNAT="Server">
<TABLE ID="tableExample" Border="1" RUNAT="Server" />
</FORM>

```

- 6 Add the `Page_Load` function.
- 7 Create and initialize a variable to be a counter for the rows.

- 8 Create and initialize a variable for the number of cells and a variable for the number of rows.
- 9 Create and initialize an array for the contents of the table.

When working with the `HTMLTable` control, you need to work with a couple of other controls, including the object that represents a cell, the `HTMLTableCell` control, and an object that represents a row, the `HTMLTableRow` control. It is best to work with these objects as collections. To do this, you can use their respective collection objects, the `HTMLTableCellCollection` and `HTMLTableRowCollection` objects.

You can build out a table from an array while the page is loading. You can use the `HTMLTable`, `HTMLTableCell`, and `HTMLTableRow` controls to accomplish this.

**Extra**

You can specify the cell padding, cell spacing, border color, and background color for the table.

**Example:**

```
<FORM RUNAT="Server">
<TABLE ID="tableExample" Border="1"
CellPadding="5" CellSpacing="0"
BorderColor="Black" BgColor="#C0C0C0"
RUNAT="Server" />
</FORM>
```

You can specify a number of properties for the table cells by setting the horizontal and vertical alignment of the cells.

**Example:**

```
HtmlTableCell cell = new HtmlTableCell();
cell.Align="Right";
cell.VAlign="Top";
```

You can use table row properties to format a table with alternating colors for each row.

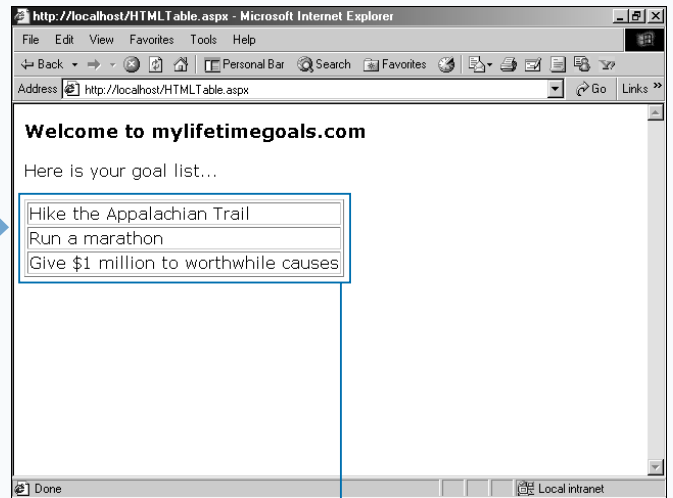
**Example:**

```
HtmlTableRow row = new HtmlTableRow();
if (rowcount%2 == 1)
row.BgColor="#C0C0C0";
rowcount++;
```

```
void Page_Load(Object sender, EventArgs e) {
    int rowcount = 0;
    int numRows = 3;
    int numcells = 1;
    string[] sGoalList = new string[3];
    sGoalList[0] = "Hike the Appalachian Trail";
    sGoalList[1] = "Run a marathon";
    sGoalList[2] = "Give $1 million to worthwhile causes";

    for (int j=0; j<numrows; j++) {
        HtmlTableRow row = new HtmlTableRow();
        rowcount++;

        for (int i=0; i<numcells; i++) {
            HtmlTableCell cell = new HtmlTableCell();
            cell.Controls.Add(new
            LiteralControl(sGoalList[j].ToString()));
            row.Cells.Add(cell);
        }
        tableExample.Rows.Add(row);
    }
}
```



**10** Process each row with a `for` loop.

**11** Create a new `HtmlTableRow` for each row.

**12** Process each cell in the row with a `for` loop.

**13** Create a new `HtmlTableCell` for each cell and add the value in the array to the cell.

**14** Add the cell to the row.

**15** Add the row to the table.

**16** Save the file and request it from the Web server.

■ The ASP.NET Web page appears.

*Note: See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.*

■ A table appears with the contents of your array.

# STORE HIDDEN INFORMATION ON A FORM

You can store information on your forms with a hidden variable. Users cannot see the data in hidden variables. Because users can view the source of the HTML to see the hidden variables, you should not use this technique with sensitive or secure information. Also, because the data is being posted back and forth between the Web browser and the Web server, you do not want to place too much information in hidden variables.

HTMLInputHidden controls are typically used for retaining state from one page to the next page. You can find that sometimes storing data in

HTMLInputHidden controls is a good alternative to managing state in Session variables. See page 226 for details on managing state with Session variables. You should store important state into HTMLInputHidden controls when the user clicks the submit button on the form.

For example, you can create a page that posts back to itself and simply moves the data from the text box and puts it into a hidden variable. When this is done, you can check for the hidden variable by viewing the source for the Web page.

## STORE HIDDEN INFORMATION ON A FORM

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void SubmitBtn_Click(object Source, EventArgs e) {
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
What suggestions do you have for improving our site?

<FORM RUNAT="Server">
<P>
<INPUT ID="inputSuggestions" TYPE="Text" SIZE="40" RUNAT="Server">
<INPUT ID="hiddenSuggestions" TYPE="Hidden" RUNAT="Server">
<INPUT TYPE="Submit" VALUE="Submit" OnServerClick="SubmitBtn_Click"
RUNAT="Server">
<P>
<SPAN ID="spanMessage" RUNAT="Server" />
</FORM>
  
```

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void SubmitBtn_Click(object Source, EventArgs e) {
    hiddenSuggestions.Value=inputSuggestions.Value;
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
What suggestions do you have for improving our site?

<FORM RUNAT="Server">
<P>
<INPUT ID="inputSuggestions" TYPE="Text" SIZE="40" RUNAT="Server">
<INPUT ID="hiddenSuggestions" TYPE="Hidden" RUNAT="Server">
<INPUT TYPE="Submit" VALUE="Submit" OnServerClick="SubmitBtn_Click"
RUNAT="Server">
<P>
<SPAN ID="spanMessage" RUNAT="Server" />
</FORM>
  
```

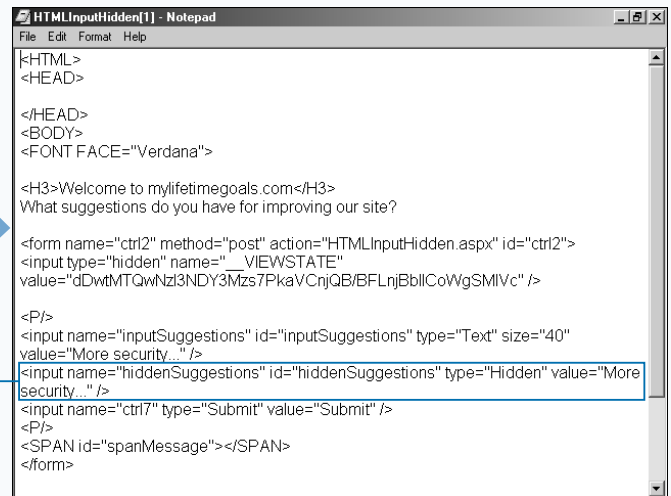
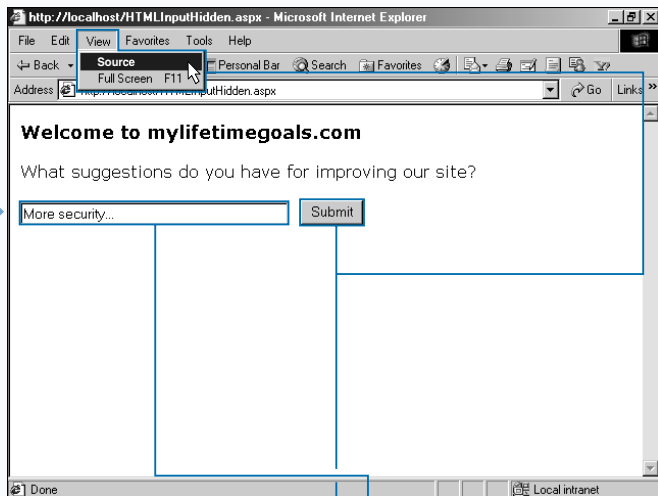
- 1 Open the `SuggestionsTemplate.aspx` template from the Code Templates directory.
- 2 Add an `HTMLTextBox` control to the form.
- 3 Add an `HTMLInputHidden` control to the form.

- 4 Set the hidden value of the variable to what was entered into the text box.

**Extra**

When you viewed the source, you may have noticed that there was another hidden variable on the form that you did not create. This hidden variable is used for state information for controls on the form. ASP.NET automatically creates this variable.

If you ever want to programmatically remove an input tag from the page that the user receives, you can set the `Visibility` property equal to `false`. This will remove the tag from the response to the user, even though the `.aspx` file has an `<input>` tag marked up in the document.



**5** Save the file and request it from the Web server.

■ The ASP.NET Web page appears.

**6** Type in a suggestion.

**7** Click the Submit button.

**8** Click View ⇌ Source from the menu in Internet Explorer.

■ The source for the page appears and the `hiddenSuggestions` tag contains the suggestion.

*Note: See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.*

# UPLOAD FILES

People who visit your Web sites can upload files to your Web server using the `HTMLInputFile` control. They can choose a local file, and then have that file uploaded to the Web server. For some applications, having the user upload a file of a predetermined format is a more convenient way of inputting data, as opposed to having the user type all of the information into a form.

To be able to send a file, you are required to modify the `HTMLForm` control to contain the attribute name/value pair of `ENCTYPE="multipart/form-data"`. This attribute instructs the Web

browser that one of the controls is a file that needs to be uploaded to the server. On the form, you can then add an `HTMLInputFile` control by using the `<input>` tag and setting the attribute's `TYPE="File"` and `RUNAT="Server"`.

After a user selects a file by using the `HTMLInputFile` control, the user will submit the form to the server, sending the file up with the HTTP request. To place the file on the server's file system, you need to write some code that will check for the file, save the file if available, and do some exception handling if there is a problem.

## UPLOAD FILES

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
    void SubmitBtn_Click(object Source, EventArgs e) {
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
What suggestions do you have for improving our site?

<FORM ENCTYPE="multipart/form-data" RUNAT="Server">
Select a file with suggestions: <INPUT ID="fileSuggestions" TYPE="File"
RUNAT="Server">
<P/>
<SPAN ID="spanMessage" RUNAT="Server"/>
<P/>
<INPUT TYPE="Submit" VALUE="Submit" OnServerClick="SubmitBtn_Click"
RUNAT="Server">
</FORM>
  
```

**1** Open the `SuggestionsTemplate.aspx` template from the Code Templates directory.

**2** Add the `ENCTYPE` attribute and specify that the form will be posting multipart form data.

**3** Add a message and an `HTMLInputFile` control to the form.

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
    void SubmitBtn_Click(object Source, EventArgs e) {
        if (fileSuggestions.PostedFile != null) {
            try {
                fileSuggestions.PostedFile.SaveAs("c:\\temp\\suggestions.bt");
                spanMessage.InnerHtml = "File uploaded successfully to" +
                    "c:\\temp\\suggestions.bt";
            }
            catch (Exception exc) {
                spanMessage.InnerHtml = "Error saving file" +
                    "c:\\temp\\suggestions.bt"+ exc.ToString();
            }
        }
    }
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
What suggestions do you have for improving our site?
  
```

**4** In the `SubmitBtn_Click` function, check to make sure there is a filename entered.

**5** Post the file up to the Web server and set the message for the `<SPAN>` tag.

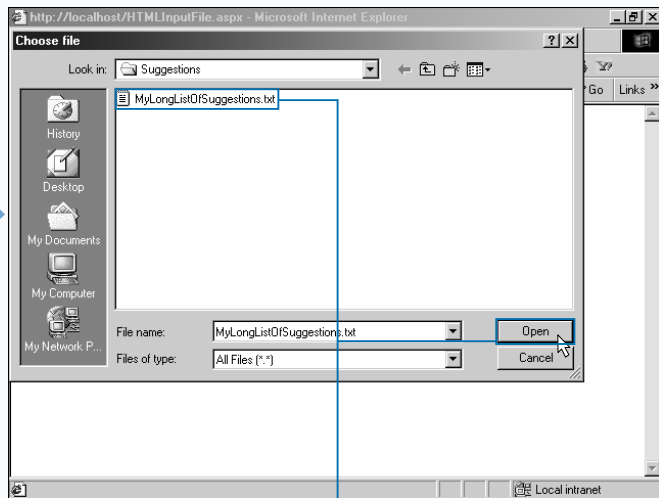
**6** Catch the exception if the post was unsuccessful and set the message to reflect an error.



## Apply It

Because there is a chance that users could overwrite files on the Web server when uploading files, you may want to make sure the names of files are unique. One way of doing this is by using the `System.Guid` class. The following modifications to the `SubmitBtn_Click` function would save files to the Temporary directory using a GUID, which stands for a Globally Unique Identifier. See the complete source code on the CD-ROM, Chapter04\Code\HTMLInputFile\_ai.aspx.

```
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void SubmitBtn_Click(object Source, EventArgs e) {
    if (fileSuggestions.PostedFile != null) {
        string stringFileName = System.Guid.NewGuid().ToString();
        try {
            fileSuggestions.PostedFile.SaveAs("c:\\temp\\" + stringFileName + ".txt");
            spanMessage.InnerHtml = "File uploaded successfully to " +
                "c:\\temp\\" + stringFileName + ".txt";
        }
        catch (Exception exc) {
            spanMessage.InnerHtml = "Error saving file" +
                "c:\\temp\\" + stringFileName + ".txt" + exc.ToString();
        }
    }
}
```



**7** Save the file and request it from the Web server.

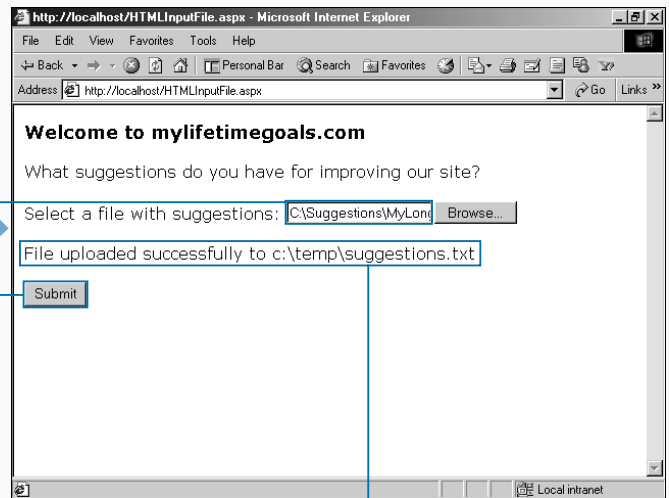
*Note: See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.*

**8** Click the Browse button.

**9** The Choose file dialog box appears.

**10** Select a file to upload to the Web server.

**11** Click Open.



**12** The path and filename appear in the text box.

**13** Click the Submit button.

**14** A message appears notifying you that the file successfully uploaded to the Web server.

# INTRODUCTION TO WEB CONTROLS

ASP.NET has provided expanded capabilities over the HTML server controls with the Web server controls. There is overlap in coverage for these

controls, but the way you program with these controls is unified across these controls.

## Basics of Web Form Controls

*Web server controls* are a close cousin to the HTML server controls covered in Chapter 4. At first glance, you may not understand why Web form controls (also known as Web controls) exist and you may get the two confused. Just like HTML controls, Web controls are HTML elements that can be processed on the server. This processing can occur before sending the Web page to the user and/or when the page is posted back to the server. Unlike HTML controls, Web controls do not always map directly to an HTML element. Also, a Web control's properties are not identical to the representative HTML element's attribute.

Seeing the high-level differences and similarities between Web server controls and HTML server controls (especially the overlap of representation for HTML elements), you should ask, "Why Web controls?" The answer is "uniformity and ease of use." Web server controls use consistent naming conventions and programming models across the controls in the namespace for Web controls. The Web controls wrap common HTML elements and give a consistent interface (hiding the programmers from the inconsistencies that cause a longer learning curve for Web development).

Web controls are defined in the `System.Web.UI.WebControls` namespace. Web forms have a slightly different convention than HTML

controls. Web controls all have the ASP namespace prefix in front of every control in the `System.Web.UI.WebControls` namespace. Like an HTML control, each tag needs to contain the `RUNAT="Server"` attribute within it. You also want to give each Web control a unique `ID` attribute, so that you have a way to reference the control in your server-side code.

You can set attributes for the Web controls to set properties of the control and handle events. Property attributes configure how the control will appear and behave as an HTML element on a Web page. As for handling events, you mark up events as attributes on a Web control. This gives you the ability to map a Web control's event to a procedure that will process the event. For example, when working with the `ASP:ImageButton` control, you can add an attribute `onServerClick` to map to a function that is called when the user clicks the button.

When working with Web controls, you may find the syntax foreign if you have not previously worked with XML. The requirement for the notation is similar to building well-formed XML documents (like closing all tags). For all HTML server controls, you should make sure that the HTML tags are properly closed and cleanly nested (overlapping tags are not allowed).

## Working with Web Controls

The `System.Web.UI.WebControls` namespace has a few controls that enable you to have better control over your HTML page display. Because you can systematically create many of these controls when the page is loading, you can work with the `ASP:Placeholder` control to specify where on the Web page the controls should be placed. Another feature that controls the layout of a Web page is the `ASP:Panel` control. This control acts as a container for other controls and can be shown and hidden using code.

## Classifications of Web Controls

Web controls overlap with the HTML controls by representing many of the same HTML elements that are available with HTML controls, but Web controls also have a list of very rich controls like the `ASP:Calendar` control. These rich controls are usually made up of many HTML elements that can collectively render the rich control. There are many Web controls to choose from and they can be classified as *Basic*, *Rich*, *List*, *Data List*, and *Validation* controls. The following tables summarize these controls:

## WEB CONTROLS

## List Web Controls

CLASS	BRIEF DESCRIPTION
CheckBoxList	Multiselection check box group.
DropDownList	Drop-down list which allows the user to select a single item.
ListBox	List box control that allows single or multiple item selection.
RadioButtonList	List control that encapsulates a group of radio button controls.

## Data List Web Controls

CLASS	BRIEF DESCRIPTION
DataGrid	Data-bound list control that displays the items from a data source in a table.
DataList	Data-bound list control that displays items using templates.

## Rich Web Controls

CLASS	BRIEF DESCRIPTION
AdRotator	Displays an advertisement banner on a Web page.
Calendar	Displays a single month calendar that enables the user to select dates and move to the next or previous month.

## Basic Web Controls

CLASS	DESCRIPTION	HTML EQUIVALENT
Button	Push button control.	<button></button>
CheckBox	Check box that enables the user to select a true or false condition.	<input type="checkbox"/>
RadioButton	Radio button control (derived from CheckBox class).	<input type="radio"/>
HyperLink	Hyperlink used to link to another resource.	<a></a>
Image	Image.	<img></img>
ImageButton	Image that responds to mouse clicks (derived from Image).	<input type="image"/>
Label	Label control.	<span></span>
LinkButton	Displays a hyperlink style button control on a Web page.	<a><img/></a>
Panel	Represents a control that acts as a container for other controls.	<div></div>
Table	Constructs a table and defines its properties.	<table></table>
TableCell	Represents a cell in a Table control.	<td></td>
TableRow	Represents a row in a Table control.	<tr></tr>
TextBox	Constructs a text box and defines its properties.	<input type="text"/> or <input type="password"/> or <textarea></textarea>

# CREATE A BUTTON FOR POSTING DATA

You can place a control on your Web forms for users to click in order to submit a form to the Web server for processing. Developers use button controls most frequently for this purpose. This chapter also looks at the `LinkButton` Web control and the `ImageButton` control, which you can use for the same purpose.

Buttons are the natural choice for a control that needs to submit form data back to a server. The `ASP:Button` control, `<ASP:BUTTON ID="cmdButton" TEXT="Continue" RUNAT="Server" />`, generates an HTML Input Submit Button tag, `<input type="submit" name="cmdButton" value="Continue"`

`id="cmdButton" />`, in the client's browser. With this Web server control, you have all the capabilities of the standard HTML input submit button plus the extended properties and state management that is available for server controls.

This section demonstrates how to create a form that uses the `Button` Web control to forward users to the second step in a process. This section uses client-side code to create a rollover behavior for the button. When the user clicks the button, the browser sends a message letting the user know the process is continuing to the next step. At this point, you can redirect the user to the next page in the process.

## CREATE A BUTTON FOR POSTING DATA

```

Untitled - Notepad
File Edit Format Help
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Button_OnClick(object Source, EventArgs e) {
    labelMessage.Text="You are continuing to Step 2...";
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Are you ready to set your goals? Click on the Continue button to go to Step 2.

<FORM RUNAT="Server">
<P>
<ASP:BUTTON ID="buttonContinue" RUNAT="Server"/>
<P>
<ASP:LABEL ID="labelMessage" RUNAT="Server" />
</FORM>

</FONT>
</BODY>
</HTML>
  
```

```

Untitled - Notepad
File Edit Format Help
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Button_OnClick(object Source, EventArgs e) {
    labelMessage.Text="You are continuing to Step 2...";
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Are you ready to set your goals? Click on the Continue button to go to Step 2.

<FORM RUNAT="Server">
<P>
<ASP:BUTTON ID="buttonContinue"
    TEXT="Continue"
    STYLE="background-color:White;border-color:black;height=30;width:100"
    RUNAT="Server"/>
<P>
<ASP:LABEL ID="labelMessage" RUNAT="Server" />
</FORM>

</FONT>
  
```

**1** Open the `WebWelcomeTemplate.aspx` template from the Code Templates directory.

**2** Add a `Button` Web control on the page for the user to click.

**3** Set the `TEXT` attribute equal to `Continue`.

**4** Set an initial style for the button.

**Extra**

The `Button` Web control has a couple of additional properties that you can use to handle some of the programming problems you have with standard HTML 4.0 Buttons. If you have more than one button, the `CommandName` and `Argument` are useful attributes that can store information that is associated with the button and can be used in the event handler.

**Example:**

```
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Button_OnClick(object Source, EventArgs e)
{
    labelMessage.Text="You are continuing to Step 2..."
    + "</BR>" + "Your Command Name was "
    + buttonContinue.CommandName + ".</BR>" +
    "Your Command Argument was "
    + buttonContinue.CommandArgument + ".</BR>";
}
</SCRIPT>
<ASP:BUTTON ID="buttonContinue"
    TEXT="Continue"
    STYLE="height=30;width:100"
    onMouseOver="this.style.backgroundColor='Silver'"
    onMouseOut="this.style.backgroundColor='White'"
    onClick="Button_OnClick"
    CommandName="Move"
    CommandArgument="Homepage"
    RUNAT="Server" />
```

```
Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Button_OnClick(object Source, EventArgs e) {
    labelMessage.Text="You are continuing to Step 2...";
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">

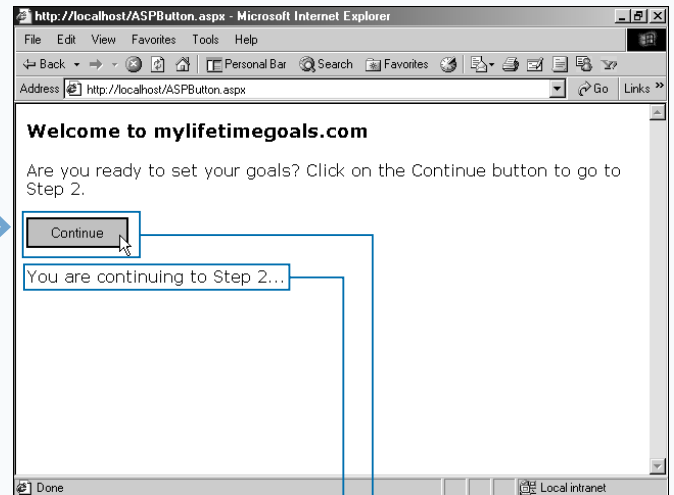
<H3>Welcome to mylifetimagoals.com</H3>
Are you ready to set your goals? Click on the Continue button to go to Step 2.

<FORM RUNAT="Server">
<P>
<ASP:BUTTON ID="buttonContinue"
    TEXT="Continue"
    STYLE="background-color:White;border-color:black;height=30;width:100"
    onMouseOver="this.style.backgroundColor='Silver'"
    onMouseOut="this.style.backgroundColor='White'"
    onClick="Button_OnClick"
    RUNAT="Server"/>
</P>
```

**5** Add the code for the `onMouseOver` event to set the background color of the button to silver.

**6** Add the code for the `onMouseOut` event to set the background color for the button back to white.

**7** Add the `onClick` event to call the `Button_OnClick` function.



**8** Save the file and request it from the Web server.

**9** Click the Continue button. A message appears.

*Note: See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.*

# CREATE A HYPERLINKED BUTTON

Like the `Button` Web control, you can use the `LinkButton` Web control to give your users a way to submit forms. This control looks like a normal hyperlink that a user can click, but it gives you the ability to process the form data by calling a subroutine when the `onClick` event is detected. This subroutine will run before the user is sent to the page designated in the link.

The `LinkButton` Web control takes an anchor tag, `<a>`, and extends its capabilities to function like an HTML input submit button. ASP 3.0 requires programmers to complete extra work to take anchor tags and enable them to submit forms. In ASP.NET, this capability is built in.

The `LinkButton` Web control resides on a server form on your ASP.NET page. The `LinkButton` control is created with the `<ASP:LINKBUTTON>` tag. Remember that you need to add an `ID` attribute and set the `RUNAT` attribute to `'Server'`. To handle the event when the user clicks the link, you should add an attribute for the `onClick` event. For the function that you call in the `onClick` event, create the code that processes the form data. When you finish processing the form, you can redirect the user to another page.

You can use the `LinkButton` control on a page that lets the user move to a second step in a process. To do this, you must open a template file and declare the `LinkButton` control and set the appropriate attributes and event handlers.

## CREATE A HYPERLINKED BUTTON

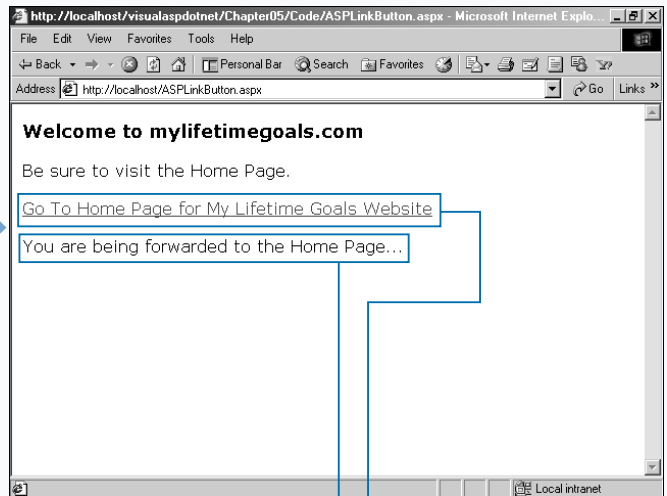
```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Button_OnClick(object Source, EventArgs e) {
    labelMessage.Text = "You are being forwarded to the Home Page...";
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Be sure to visit the Home Page

<FORM RUNAT="Server">
<P/>
<ASP:LINKBUTTON TEXT="Go To Home Page for My Lifetime Goals Website"
onClick="Button_OnClick" RUNAT="Server"/>
<P/>
<ASP:LABEL ID="labelMessage" RUNAT="Server" />
</FORM>

```



**1** Open the `WebWelcomeTemplate.aspx` template from the Code Templates directory.

**2** Add a `LinkButton` Web control to the page and have it call the `Button_OnClick` when the `onClick` event is fired.

**3** Save the file and request it from the Web server.

**4** Click the link.  
A message appears.

*Note: See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.*

# CREATE A GRAPHICAL BUTTON

**Y**ou can use an image for a button on a form to create an animated button. You can do this if you use an `ImageButton` Web control in combination with some client-side code to create a rollover effect.

The `ImageButton` Web control will reside on a server form on your ASP.NET page. The `ImageButton` control is created with the `<ASP:IMAGEBUTTON/>` tag. As with other HTML and Web controls, set the `RUNAT="Server"` attribute in the tag. To set the image for the control, set the `IMAGEURL` to the path and the filename of the image that you wish to display. To create a rollover effect,

add the `onMouseOver` and `onMouseOut` event handlers. Set `onMouseOver` to the path and filename of the image you want to display when the user puts their mouse on the image. You should use the `onMouseOut` event to set the image back to the original image path and filename when the user moves the mouse off the image.

The result of using an `ImageButton` Web control on a form will generate an `<input type="image">` tag in the user's browser. This Web control addresses a common Web programming need, the ability to use an image as a button.

## CREATE A GRAPHICAL BUTTON

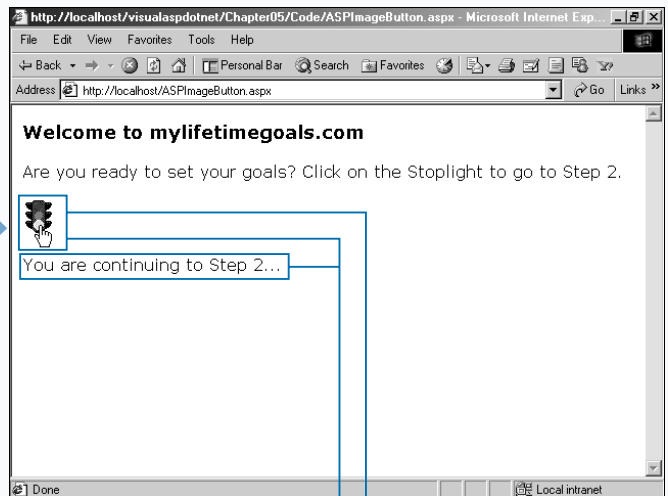
```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
    void Button_OnClick(object Source, ImageClickEventArgs e) {
        labelMessage.Text = "You are continuing to Step 2...";
    }
</SCRIPT>
</HEAD>
<BODY>
<FONT face="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Are you ready to set your goals? Click on the Stoplight to go to Step 2.

<FORM RUNAT="Server">
<P>
<ASP:IMAGEBUTTON ID="imagebuttonImage" RUNAT="Server"
IMAGEURL="images/redlight.gif" OnClick="Button_OnClick"
onMouseOver="this.src='images/greenlight.gif';"
onMouseOut="this.src='images/redlight.gif'"
/>
<P>
<ASP:LABEL ID="labelMessage" RUNAT="Server"

```



- 1 Open the `WebWelcomeTemplate.aspx` template from the Code Templates directory.
- 2 Add a `ImageButton` Web control to the page.
- 3 Set the initial value for the image.

- 4 Call the `Button_OnClick` when the `onClick` event is fired for the `ImageButton` Web control.
- 5 Add the `onMouseOver` and `onMouseOut` event handlers to create the rollover effect.

- 6 Save the file and request it from the Web server.

*Note:* See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.

- A red light appears.
- 7 Position your mouse over the stoplight and click.
- A green light appears with a continuing message.



# REQUEST TEXT INPUT

You can use the `TextBox` Web control to enable a user to type information into a form. Users can type in information such as text, numbers, and dates. You can work with a couple different types of text boxes, including a single-line text box, a multi-line text box, and a password text box.

The `TextBox` Web control resides on a server form on your ASP.NET page. You can create the `TextBox` control with the `<ASP: TEXTBOX />` tag. As with other HTML and Web controls, set the `RUNAT="Server"` attribute in the tag. The type of text box is specified by setting the `TEXTMODE` attribute. The values for `TEXTMODE` are `SingleLine`, `Multiline`, and `Password`.

You can create a simple login page to test for a static password. To do this, first add a single-line text box for the user to type his or her login name. Then, add a password text box. To enable users to submit the form, you can add a button which calls a server-side function. In that function, you can check what the user entered in the password box against a value. Depending upon success or failure of the password, you can format an appropriate message to display using a label on the form.

## REQUEST TEXT INPUT

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Please login to the secured area. You can use "goals" as a guest password.

<FORM RUNAT="Server">
<P/>
<ASP:BUTTON OnClick="SubmitBtn_Click" TEXT="Submit" RUNAT="Server"/>
</FORM>

</FONT>
</BODY>
</HTML>

```

- 1 Open the `GenericTemplate.aspx` template from the Code Templates directory.
- 2 Type a heading for the page.
- 3 Type a message to the user.

- 4 Type a Server form to the page.
- 5 Add a `Button` Web control that calls the `SubmitBtn_Click` for the `onClick` event.

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Please login to the secured area. You can use "goals" as a guest password.

<FORM RUNAT="Server">
Enter Name: <ASP:TEXTBOX ID="inputName" TEXTMODE="SingleLine" TEXT=""
WIDTH="200px" RUNAT="Server"/>
<P/>
Enter Password: <ASP:TEXTBOX ID="inputPassword" TEXTMODE="Password"
TEXT="" WIDTH="200px" RUNAT="Server"/>|
<P/>
<ASP:BUTTON OnClick="SubmitBtn_Click" TEXT="Submit" RUNAT="Server"/>
<P/>
<ASP:LABEL ID="labelMessage" style="color:red" RUNAT="Server"/>
</FORM>

</FONT>
</BODY>
</HTML>

```

- 6 Add a `Label` control for the message.
- 7 Add a single-line text box and set the `WIDTH` attribute to 200 pixels.

- 8 Add a password and set the `WIDTH` attribute to 200 pixels.



**Extra**

You can create a text box that is read only by setting the `Disabled` attribute of the `TextBox` Web control to `True`.

**Example:**

```
Enter Name: <ASP:TEXTBOX ID="inputName"
TEXTMODE="SingleLine" TEXT="" WIDTH="200px"
RUNAT="Server" DISABLED="True" />
```

You can programmatically hide the text boxes from the user by setting the `Visible` attribute of the `TextBox` Web control equal to `False`.

**Example:**

```
Enter Name: <ASP:TEXTBOX ID="inputName"
TEXTMODE="SingleLine" TEXT="" WIDTH="200px"
RUNAT="Server" VISIBLE="False" />
```

You can limit the number of characters that a user can type into a textbox. with the `MAXLENGTH` attribute.

**Example:**

```
Enter Name: <ASP:TEXTBOX ID="inputName"
TEXTMODE="SingleLine" TEXT="" WIDTH="200px"
RUNAT="Server" MAXLENGTH="40" />
```

You can create a multiple line textbox by setting the `TEXTMODE` attribute to `MultiLine`.

**Example:**

```
Enter Name: <ASP:TEXTBOX ID="inputName"
TEXTMODE="MultiLine" TEXT="" WIDTH="200px"
RUNAT="Server" MAXLENGTH="40" />
```

```

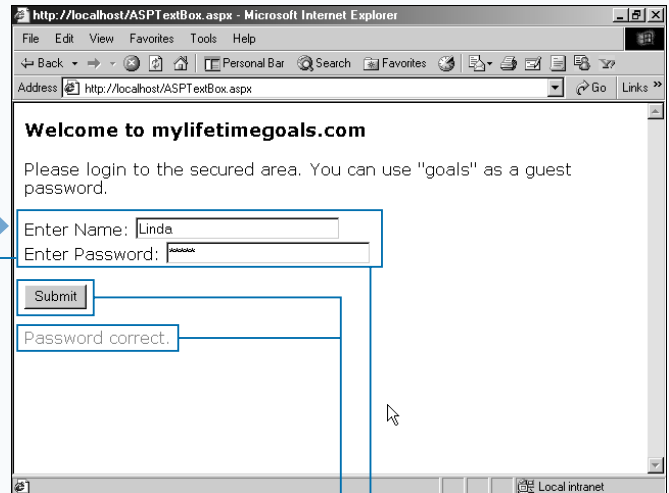
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void SubmitBtn_Click(object Source, EventArgs e){
    if (inputPassword.Text == "goals") {
        labelMessage.Text="Password correct.";
    }
    else {
        labelMessage.Text="That password is not correct.";
    }
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
Please login to the secured area. You can use "goals" as a guest password.

<FORM RUNAT="Server">
Enter Name: <ASP:TEXTBOX ID="inputName" TEXTMODE="SingleLine" TEXT=""
WIDTH="200px" RUNAT="Server"/>

```

**9** Create a function called `SubmitBtn_Click` to check the password.

**10** Add an `if` statement to check the password and display the message using the `Label` Web control on the page.



**11** Save the file and request it from the Web server.

**12** Type in a name and a correct password.

**13** Click the Submit button.

A message appears notifying you that the password is correct.

*Note: See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.*

# REQUEST BOOLEAN INPUT

Some Web page form questions that you ask your users require a yes/no or true/false response. For these types of responses, you can use the `CheckBox` Web server control. This control functions similarly to the `HTMLInputCheckBox` HTML server control.

The `CheckBox` Web control will reside on a server form on your ASP.NET page. The `CheckBox` control is created with the `<ASP: CHECKBOX >` tag. You need an `ID` attribute to give the control a name, which is how you reference it in code. As with other Web server controls, set the `RUNAT` attribute to `Server`.

Use the `TEXT` attribute to set what is displayed to users for that check box. To process the check box, you should use the `Checked` property to see whether it is set to `true`. If set to `true`, that means that the user clicked the check box.

You can use multiple check boxes on one Web form. In this case of multiple check boxes, you can use the `CheckBoxList` Web server control. For most cases, you will create a multi selection check box group dynamically by binding the control to a data source. See page 128 for how controls are data bound.

## REQUEST BOOLEAN INPUT

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
What suggestions do you have for improving our site?

<FORM RUNAT="Server">
</FORM>

</FONT>
</BODY>
</HTML>

```

**1** Open the `GenericTemplate.aspx` template from the Code Templates directory.

**2** Type a heading for the page.

**3** Type a message to the user.

**4** Add a server form to the page.

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
What suggestions do you have for improving our site?

<FORM RUNAT="Server">
<P/>
<ASP:CHECKBOX ID="checkboxSuggestions" TEXT="Have more goals to choose from." RUNAT="Server" />
<P/>
<ASP:BUTTON TEXT="Submit" OnClick="SubmitBtn_Click" RUNAT="Server"/>
<P/>
<ASP:LABEL ID="labelMessage" RUNAT="SERVER"/>
</FORM>

</FONT>
</BODY>
</HTML>

```

**5** Add a `Button` Web control that calls the `SubmitBtn_Click` for the `onClick` event.

**6** Add a `Label` control for the message.

**7** Add a `CheckBox` Web server control.

## Extra

You can use the `CheckBoxList` Web server control for working with multiple check boxes.

## Example:

```
<FORM RUNAT="Server">
<P/>
<ASP:CHECKBOXLIST ID="checkboxlistSuggestions"
RUNAT="Server">
<ASP:LISTITEM>Have more goals to choose
from.</ASP:LISTITEM>
<ASP:LISTITEM>Have more goal
categories.</ASP:LISTITEM>
<ASP:LISTITEM>Make the goal setting wizard
easier.</ASP:LISTITEM>
</ASP:CHECKBOXLIST>
<P/>
<ASP:BUTTON TEXT="Submit"
OnClick="SubmitBtn_Click" RUNAT="Server"/>
<P/>
<ASP:LABEL ID="labelMessage" RUNAT="SERVER"/>
</FORM>
```

To respond to a Web form that contains multiple check boxes, you can use the event handler code.

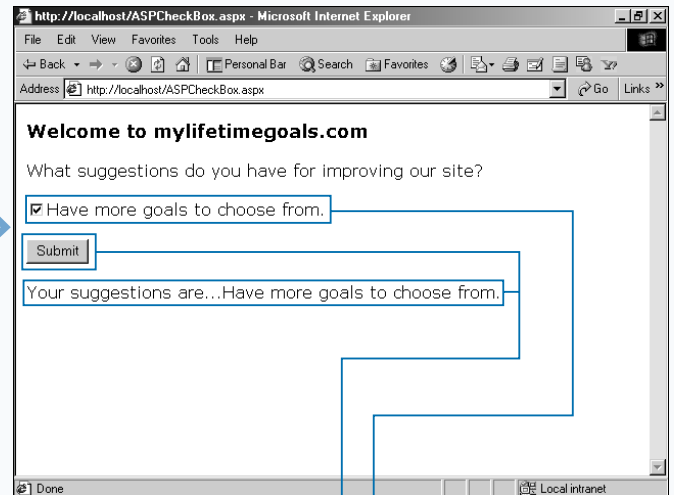
## Example:

```
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void SubmitBtn_Click(object Source,
EventArgs e) {
String sMessage = "";
for (int i=0; i <
checkboxlistSuggestions.Items.Count; i++) {
if (checkboxlistSuggestions.
Items[ i ].Selected) {
sMessage = sMessage +
checkboxlistSuggestions.Items[i].Text;
sMessage = sMessage + "<BR />";
}
}
if (sMessage != "") {
labelMessage.Text ="Your suggestions"
+ "are...<BR />" + sMessage;
}
else {
labelMessage.Text = "You have no
suggestions.";
}
}
</SCRIPT>
```

```
Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void SubmitBtn_Click(object Source, EventArgs e) {
if (checkboxSuggestions.Checked == true) {
labelMessage.Text ="Your suggestions are..." +
checkboxSuggestions.Text;
}
else {
labelMessage.Text = "You have no suggestions.";
}
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
What suggestions do you have for improving our site?
<FORM RUNAT="Server">
<P/>
<ASP:CHECKBOX ID="checkboxSuggestions" TEXT="Have more goals to choose
from."/>
```

**8** Create a function called `SubmitBtn_Click` to check the password.

**9** Add an `if` statement to check whether the control was checked and set the appropriate message.



**10** Save the file and request it from the Web server.

*Note: See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.*

**11** Click to select the check box.

**12** Click the Submit button.  
The suggestion echoes back.

# REQUEST A SELECTION FROM A GROUP

You can use radio buttons to have a user select a single option from a group of choices. The `RadioButton` control functions similarly to the `HTMLRadioButton` server control.

To employ the `RadioButton` Web server control on your server form, you must first declare a `RadioButton` Web server control with the `<ASP:RadioButton RUNAT="Server">` tag. Then, you must give each radio control button that you want a unique `ID` attribute. To associate a group of radio buttons, set the `GROUPNAME` attribute to be the same for all controls in the group. You can also specify which of the radio buttons is checked when

the user requests the page. You can do this by setting the `CHECKED` attribute for the radio button to `True`.

Radio buttons are typically implemented in groups of two or more, where you force the user to choose one option out of a list of options. To conveniently work with a group of radio control buttons, you can use the `RadioButtonList` Web server control.

In this example, you are asking a question that requires only one answer from a group of answers. To do this, you create a form with multiple `RadioButton` Web server controls in the same group. When the user submits the form, the `Checked` property of the controls formats a message.

## REQUEST A SELECTION FROM A GROUP

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
What suggestions do you have for improving our site?

<FORM RUNAT="Server">
</FORM>

</FONT>
</BODY>
</HTML>
  
```

**1** Open the `GenericTemplate.aspx` template from the Code Templates directory.

**2** Type a heading for the page.

**3** Type a message to the user.

**4** Add a server form to the page.

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
What suggestions do you have for improving our site?

<FORM RUNAT="Server">
<P/>
<ASP:RADIOBUTTON ID="radiobuttonSuggestionsLess" TEXT="Fewer Goals"
CHECKED="True" GROUPNAME="Suggestions" RUNAT="Server" /><BR/>
<ASP:RADIOBUTTON ID="radiobuttonSuggestionsMore" TEXT="More Goals"
CHECKED="False" GROUPNAME="Suggestions" RUNAT="Server" /><BR/>
<ASP:RADIOBUTTON ID="radiobuttonSuggestionsSame" TEXT="Same Number of
Goals" CHECKED="False" GROUPNAME="Suggestions" RUNAT="Server" /><BR/>
<P/>
<INPUT TYPE="Submit" VALUE="Submit" OnServerClick="SubmitBtn_Click"
RUNAT="Server">
<P/>
<ASP:LABEL ID="labelMessage" RUNAT="Server"/>
</FORM>
  
```

**5** Add a `Button` Web control that calls the `SubmitBtn_Click` for the `onClick` event.

**6** Add a `Label` control for the message.

**7** Add a `RadioButton` Web server control for each of the radio button options and set the `GROUPNAME` attribute for the controls to be the same so they are in a group.

**Extra**

You can use the `RadioButtonList` Web server control for working with multiple `RadioButton` Web server controls.

**Example:**

```
<FORM RUNAT="Server">
<P/>
<ASP:RADIOBUTTONLIST
ID="radiobuttonlistSuggestions"
RUNAT="Server">
<ASP:LISTITEM>Fewer Goals</ASP:LISTITEM>
<ASP:LISTITEM>More Goals</ASP:LISTITEM>
<ASP:LISTITEM>Same Number of
Goals</ASP:LISTITEM>
</ASP:RADIOBUTTONLIST>
<P/>
<INPUT TYPE="Submit" VALUE="Submit"
OnServerClick="SubmitBtn_Click"
RUNAT="Server">
<P/>
<ASP:LABEL ID="labelMessage" RUNAT="Server" />
</FORM>
```

By using this `RadioButtonList` Web server control, you can use the `SelectedIndex` property to find out which radio button was selected.

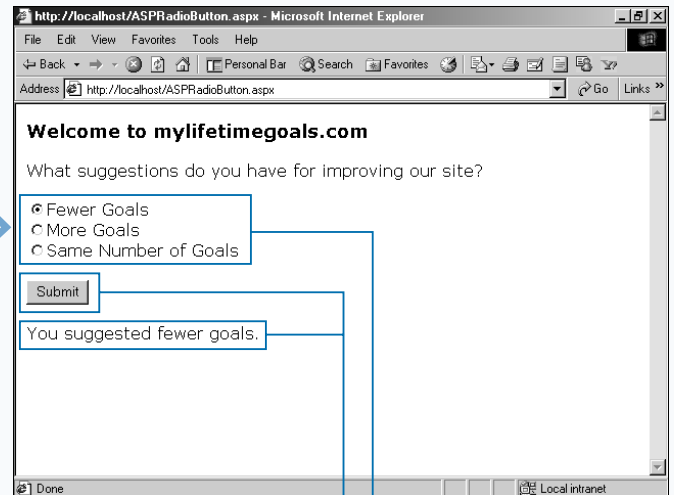
**Example:**

```
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void SubmitBtn_Click(object Source, EventArgs e)
{
    if
    (radiobuttonlistSuggestions.SelectedIndex > -1)
    {
        labelMessage.Text = "You suggested "
+ radiobuttonlistSuggestions.SelectedItem.Text;
    }
}
</SCRIPT>
```

```
Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void SubmitBtn_Click(object Source, EventArgs e) {
    if (radiobuttonSuggestionsLess.Checked == true)
        labelMessage.Text = "You suggested fewer goals.";
    else if (radiobuttonSuggestionsMore.Checked == true)
        labelMessage.Text = "You suggested more goals.";
    else if (radiobuttonSuggestionsSame.Checked == true)
        labelMessage.Text = "You suggested same number of goals.";
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
What suggestions do you have for improving our site?
<FORM RUNAT="Server">
<P/>
<ASP:RADIOBUTTON ID="radiobuttonSuggestionsLess" TEXT="Fewer Goals"
CHECKED="True" GROUPNAME="Suggestions" RUNAT="Server" /><BR/>
<ASP:RADIOBUTTON ID="radiobuttonSuggestionsMore" TEXT="More Goals"
CHECKED="False" GROUPNAME="Suggestions" RUNAT="Server" /><BR/>
<ASP:RADIOBUTTON ID="radiobuttonSuggestionsSame" TEXT="Same Number of Goals"
CHECKED="False" GROUPNAME="Suggestions" RUNAT="Server" />
</FORM>
</BODY>
</HTML>
```

**8** Create a function called `SubmitBtn_Click` to check the suggestion selected.

**9** Add an `if` statement to check whether the control was checked and send the appropriate message.



**10** Save the file and request it from the Web server.

*Note:* See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.

**11** Click to select the first radio button.

**12** Click the Submit button. The suggestion echoes back.

# REQUEST INPUT FROM A DROP-DOWN LIST

You can use a drop-down list box for soliciting input from a user where you want to give a group of options and require the user to select only one of the options. The `DropDownList` Web server control gives you a way to create this HTML control and work with the control programmatically.

The `DropDownList` Web control resides on a server form on your ASP.NET page. The `DropDownList` control is created with the `<ASP: DROPDOWNLIST >` tag. You need to add an `ID` attribute and set the `RUNAT` attribute to `Server` for the control to work properly. To present a list of options, the drop-down list box

holds each option in its own tag, `<ASP:LISTITEM>`. To process the `DropDownList` Web server control, you can use the `SelectedItem.Text` property to find the value of the option that was selected.

You can use the `DropDownList` Web server control and the `Listitem` controls to display a form for gathering input for suggestions. The `DropDownList` control is used to generate a drop-down list box on a form. When the user submits the form, the Web server reads the selected value as a property of the `DropDownList` control and echoes the suggestion back to the user.

## REQUEST INPUT FROM A DROP-DOWN LIST

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimgoals.com</H3>
What suggestions do you have for improving our site?

<FORM RUNAT="Server">
<P/>
<ASP:BUTTON TEXT="Submit" OnClick="SubmitBtn_Click" RUNAT="Server"/>
</FORM>

</FONT>
</BODY>
</HTML>
  
```

**1** Open the `GenericTemplate.aspx` template from the Code Templates directory.

**2** Type a heading for the page.

**3** Type a message to the user.

**4** Add a server form to the page.

**5** Add a `Button` Web control that calls the `SubmitBtn_Click` for the `onClick` event.

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimgoals.com</H3>
What suggestions do you have for improving our site?

<FORM RUNAT="Server">
<P/>
<ASP:DROPDOWNLIST ID="dropdownlistSuggestions" RUNAT="Server">
  <ASP:LISTITEM>Fewer Goals</ASP:LISTITEM>
  <ASP:LISTITEM>More Goals</ASP:LISTITEM>
  <ASP:LISTITEM>Same Number of Goals</ASP:LISTITEM>
</ASP:DROPDOWNLIST>
<ASP:BUTTON TEXT="Submit" OnClick="SubmitBtn_Click" RUNAT="Server"/>
<P/>
<ASP:LABEL ID="labelMessage" RUNAT="Server"/>
</FORM>

</FONT>
  
```

**6** Add a `Label` control for the message.

**7** Add a `DropDownList` Web server control and set its attributes.

**8** Add `Listitem` Web server controls for each option.

Apply It

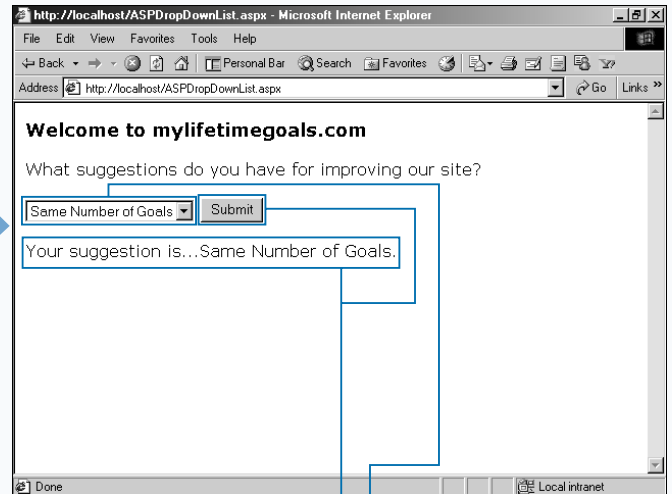
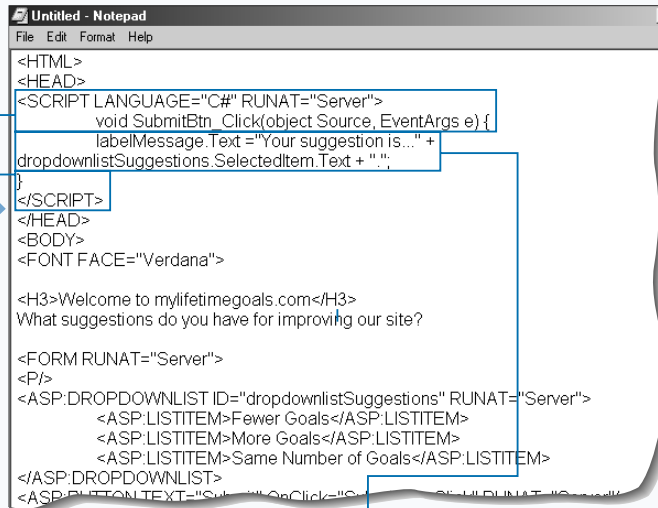
As with the rest of the Web server controls, you can data-bind the control to a data source.

TYPE THIS:

```
<HTML><HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
    void Page_Load(Object Sender, EventArgs E) {
        if (!IsPostBack) {
            ArrayList alSuggestions = new ArrayList();
            alSuggestions.Add ("Fewer Goals");
            alSuggestions.Add ("More Goals");
            alSuggestions.Add ("Same Number of Goals");
            dropdownlistSuggestions.DataSource = alSuggestions;
            dropdownlistSuggestions.DataBind();
        }
    }
    void SubmitBtn_Click(object Source, EventArgs e) {
        labelMessage.Text ="Your suggestion is..."
        + dropdownlistSuggestions.SelectedItem.Text + ".";
    }
</SCRIPT></HEAD><BODY>
    <FORM RUNAT="Server" ID="Form1">
        <P /><ASP:DROPDOWNLIST ID="dropdownlistSuggestions"
            RUNAT="Server" />
        <ASP:BUTTON TEXT="Submit" OnClick="SubmitBtn_Click"
            RUNAT="Server" ID="Button1" /><P />
        <ASP:LABEL ID="labelMessage" RUNAT="Server" />
    </FORM></BODY>
</HTML>
```

RESULT:

A drop-down list appears with the three suggested goals and a submit button.



9 Create a function called `SubmitBtn_Click` to check the input.

10 Set the label according to the input received from the user.

11 Save the file and request it from the Web server.

*Note: See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.*

12 Click  to select an option.

13 Click the Submit button.

The suggestion echoes back on screen.

# REQUEST DATES FROM A CALENDAR

You can use the `Calendar` Web server control when you need a user to select a valid date. This is one of the richest Web server controls available in the ASP.NET Framework. You can have users select different ranges for dates as well, including weeks and months. The `Calendar` control can be generated using only HTML, so the control works with most browser types.

The `Calendar` Web control will reside on a server form on your ASP.NET page. The `Calendar` control is created with the `<ASP: CALENDAR>` tag. You need to add an `ID` attribute and set the `RUNAT` attribute to `Server` for the control to run properly. To determine which date a user selects, you can write an event

handler for the `SelectionChanged` event. In the event handling code, you should check for the `SelectedDate` property to retrieve the date that the user chooses.

The `Calendar` control supports four date selection modes. You can select any single day with the `Day` mode. If you want to give the option for selecting single day or a week at a time, you can set the mode to `DayWeek`. You can give your users all the section options (select a day, week, or month) by setting the mode to `DayWeekMonth`. The last option, `None`, will disable selection of the calendar.

## REQUEST DATES FROM A CALENDAR

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Select the date you wish to accomplish this goal by.

<FORM RUNAT="Server">
</FORM>

</FONT>
</BODY>
</HTML>

```

**1** Open the `GenericTemplate.aspx` template from the Code Templates directory.

**2** Type a heading for the page.

**3** Type a message to the user.

**4** Add a server form to the page.

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Select the date you wish to accomplish this goal by.

<FORM RUNAT="Server">
<ASP:CALENDAR ID="calendarGoal" onSelectionChanged="Date_Selected"
RUNAT="Server" />
</FORM>
<P/>
<ASP:LABEL ID="labelMessage" RUNAT="Server" />
</FORM>

</FONT>
</BODY>
</HTML>

```

**5** Add a `Label` control for the message.

**6** Add a `Calendar` control to the page and have the control call the `Date_Selected` function for the `onSelectionChanged` event.



**Extra**

You can set a number of properties for the Calendar control to customize the control. The following code can customize the control in a number of ways, such as enabling the user to select a week or a month at a time as opposed to just a single date.

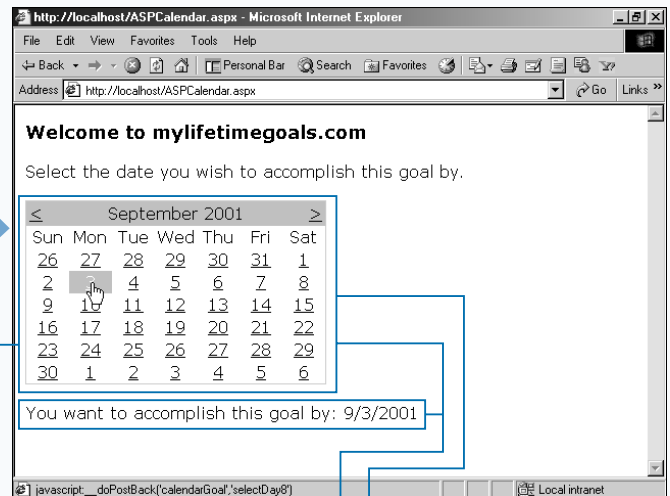
**Example:**

```
<ASP:CALENDAR ID="calendarGoal" RUNAT="Server"
onSelectionChanged="Date_Selected"
DAYNAMEFORMAT="FirstLetter"
SELECTIONMODE="DayWeekMonth"
FONT-NAME="Verdana" FONT-SIZE="12px"
HEIGHT="180px" WIDTH="230px"
TODAYDAYSTYLE-FONT-BOLD="True"
DAYHEADERSTYLE-FONT-BOLD="True"
OTHERMONTHDAYSTYLE-FORECOLOR="Gray"
TITLESTYLE-BACKCOLOR="Navy"
TITLESTYLE-FORECOLOR="White"
TITLESTYLE-FONT-BOLD="True"
SELECTEDDAYSTYLE-BACKCOLOR="LightBlue"
SELECTEDDAYSTYLE-FONT-BOLD="True"
```

```
Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Date_Selected(object s, EventArgs e) {
    labelMessage.Text = "You want to accomplish this goal by: " +
    calendarGoal.SelectedDate.ToShortDateString();
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3>Welcome to mylifetimagoals.com</H3>
Select the date you wish to accomplish this goal by.
<FORM RUNAT="Server">
<ASP:CALENDAR ID="calendarGoal" onSelectionChanged="Date_Selected"
RUNAT="Server" />
<P>
<ASP:LABEL ID="labelMessage" RUNAT="Server" />
</FORM>
```

**7** Create the `Date_Selected` function.

**8** Set the label according to the input received from the user.



**9** Save the file and request it from the Web server.

**10** A calendar appears.

**11** Click to select a date.

**12** The date echoes back.

*Note: See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.*

# CREATE A LINK

To create a hyperlink programmatically with Web server controls, you can use the `HyperLink` control. You can use this control to set properties for the hyperlink, including where the user is directed to when clicking the hyperlink and what text represents the hyperlink.

If you want to create a link programmatically, you can use the `Page_Load` event to set the hyperlink's properties. To create a `HyperLink` Web server control on an ASP.NET page, you must add the `<ASP:HYPERLINK RUNAT="Server">` tag to the page. You should also add an `ID` attribute for the control so that you can reference it in your code.

The `HREF` for the anchor tag represents the URL that the user gets sent to after clicking the anchor tag. The anchor tag surrounds the text, or HTML, that is hyperlinked (underlined) in the user's Web browser.

The following example lets you use the `Page_Load` event to set the properties for the `HyperLink` Web server control. You must set the `HREF` and the `TEXT` properties. When you request the ASP.NET Web page from the server, the `Page_Load` event fires and the server creates the anchor tag using the code in the server-side event.

## CREATE A LINK

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Page_Load(Object sender, EventArgs e) {
    hyperlinkHomePage.NavigateUrl = "http://www.mylifetimegoals.com";
    hyperlinkHomePage.Text = "Go To Home Page for My Lifetime Goals
Website";
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Be sure to visit the Home Page.

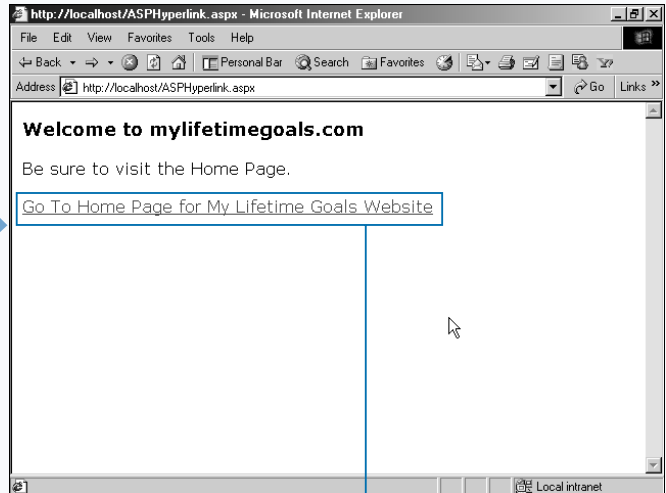
<FORM RUNAT="Server">
<P/>
<ASP:HYPERLINK ID="hyperlinkHomePage" RUNAT="Server"/>
</FORM>

</FONT>
  
```

**1** Open the `WebVisitHomepageLinkTemplate.aspx` template from the Code Templates directory.

**2** Add a `HyperLink` Web server control to the form.

**3** Add the code to set the `HREF` and `Text` properties for the `HyperLink` control.



**4** Save the file and request it from the Web server.

The page appears with a `HyperLink`.

*Note: See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.*

# RENDER AN IMAGE

You can use the `Image` Web server control to work with your images. For example, you can use the `Image` control to specify what image appears when the page first displays. Then, through code, you can easily replace this image.

The `Image` Web server control functions very similarly to the `HTMLImage` control. The `Image` Web control will reside on a server form on your ASP.NET page. The `Image` control is created with the `<ASP: IMAGE >` tag. You will then need to add an `ID` attribute and set the `RUNAT` attribute to `Server`. You can then add code on your page that modifies this control. You can

set the `IMAGEURL` attribute for mapping the path of the image source file.

The `Image` Web server control does not support any events, which separates it from most of the other Web server controls. One attribute that is useful is the `ALTERNATETEXT` attribute. This displays if the user's Web browser has graphics turned off or the source for the graphic is missing. This text for this attribute will also show as pop-up help text (if the user holds the mouse pointer over the image). You can make the pop-up help text different than the alternative text by using the `TOOLTIP` attribute.

## RENDER AN IMAGE

```

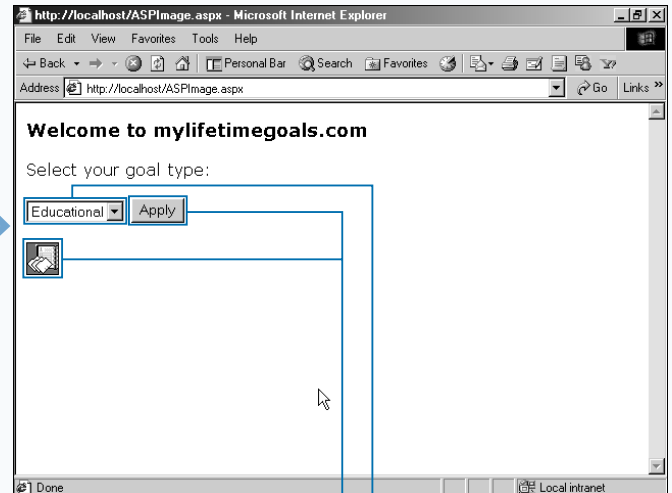
Untitled - Notepad
File Edit Format Help
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void SubmitButton_Click(Object Sender, EventArgs e) {
    imageGoalImages.ImageUrl = "images/" +
    dropdownlistGoalImages.SelectedItem.Value;
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Select your goal type:

<FORM RUNAT="Server">
<ASP:DROPDOWNLIST ID="dropdownlistGoalImages" RUNAT="Server">
    <ASP:LISTITEM VALUE="travel.gif">Travel</ASP:LISTITEM>
    <ASP:LISTITEM VALUE="career.gif">Career</ASP:LISTITEM>
    <ASP:LISTITEM VALUE="educational.gif">Educational</ASP:LISTITEM>
</ASP:DROPDOWNLIST>
<ASP:BUTTON TEXT="Apply" ONCLICK="SubmitButton_Click" RUNAT="Server"/>
</P>
<ASP:IMAGE ID="imageGoalImages" IMAGEURL="images/travel.gif"
ALTERNATETEXT="Travel" RUNAT="Server" />
  
```

**1** Open the `WebGoalTypesTemplate.aspx` template from the Code Templates directory.

**2** Add an `Image` control to the server form.



**3** Save the file and request it from the Web server.

*Note:* See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.

**4** Click to select the Educational goal type from the drop-down list box.

**5** Click the Apply button.

The image is updated with the Education image.

The page appears with the Travel image.

# BUILD A TABLE

You can use the `Table` Web server control to dynamically create and modify tables in your ASP pages. You can also use the `Table` Web server control to set table properties such as background color, cell spacing and padding, and border size and color. You can use the HTML server control `HTMLTable` for the same purpose as this control.

You can populate the table in one of the events, such as `Page_Load`, before the page is generated. The `Table` Web control will reside on a server form on your ASP.NET page. The `Table` control is created with the `<ASP: Table>` tag. You need to add an `ID` attribute and set the `RUNAT` attribute to `Server` for the control to work properly.

When working with the `Table` Web control, you can work with several other controls. This includes the object that represents a cell, the `TableCell` control, and an object that represents a row, the `TableRow` control.

Because you normally work with these members of the `TableCell` and `TableRow` controls as collections, you can use their respective collection objects, the `TableCellCollection` and `TableRowCollection` objects. With these collections, you can create an iterative statement that will loop through all members of the collections and make programmatic modifications to each item in the table.

## BUILD A TABLE

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Here is your goal list...

<P/>
<FORM RUNAT="Server">
<ASP:TABLE ID="tableExample" Border="1" RUNAT="Server" />
</FORM>

</FONT>
</BODY>
</HTML>

```

- 1 Open the `GenericTemplate.aspx` template from the Code Templates directory.
- 2 Add a heading for the page.
- 3 Add a message to the user.

- 4 Add a form to the page.
- 5 Add a table to the page and set the `BORDER` attribute equal to 1.

*Note: You need to give the form an `ID` attribute and set the `RUNAT` attribute to "Server".*

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Page_Load(Object sender, EventArgs e) {
    int rowcount = 0;
    int numRows = 3;
    int numcells = 1;
    string[] sGoalList = new string[3];
    sGoalList[0] = "Hike the Appalachian Trail";
    sGoalList[1] = "Run a marathon";
    sGoalList[2] = "Give $1 million to worthwhile causes";
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>

```

- 6 Add the `Page_Load` function.
- 7 Create and initialize a variable for counting rows.

- 8 Create and initialize variables for number of rows and number of cells.
- 9 Create and initialize an array for the contents of the table.

## Apply It

Not only can you add literals to a table cell, but you can also add other controls. The full page for this example is available on the CD.

### TYPE THIS:

```
void Page_Load(Object sender, EventArgs e) {
    string[] sGoalList = {"Hike the Appalachian Trail",
        "Give $1 million to worthwhile causes"};
    int iRows = sGoalList.GetUpperBound(0) + 1;

    for (int i=0; i<iRows; i++) {
        TableRow trGoals = new TableRow();
        TableCell tcGoals = new TableCell();
        tcGoals.Controls.Add
            (new
                LiteralControl(sGoalList[i]));
        trGoals.Cells.Add(tcGoals);
        TableCell tcGoal2s = new
            TableCell();
        tcGoal2s.Controls.Add(new
            CheckBox());
        trGoals.Cells.Add(tcGoal2s);
        Table1.Rows.Add(trGoals);
    }
}
```

### RESULT:

A table that contains a list of goals with a checkbox next to each.

```
Untitled - Notepad
File Edit Format Help

string[] sGoalList = new string[3];
sGoalList[0] = "Hike the Appalachian Trail";
sGoalList[1] = "Run a marathon";
sGoalList[2] = "Give $1 million to worthwhile causes";

for (int j=0; j<numrows; j++) {
    TableRow row = new TableRow();
    rowcount++;

    for (int i=0; i<numcells; i++) {
        TableCell cell = new TableCell();
        cell.Controls.Add(new
            LiteralControl(sGoalList[j].ToString()));
        row.Cells.Add(cell);
    }
    tableExample.Rows.Add(row);
}
```

**10** Process each row with a `for` loop.

**11** Create a new `HTMLTableRow` for each row.

**12** Process each cell in the row with a `for` loop.

**13** Create a new `HTMLTableCell` for each cell and add the value in the array to the cell.

**14** Add the cell to the row.

**15** Add the row to the table.

```
http://localhost/ASPTable.aspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Home Personal Bar Search Favorites
Address http://localhost/ASPTable.aspx Go Links
Welcome to mylifetimegoals.com
Here is your goal list...
Hike the Appalachian Trail
Run a marathon
Give $1 million to worthwhile causes
Done Local intranet
```

**16** Save the file and request it from the Web server.

A table with the contents of your array appears.

*Note:* See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.

# MANIPULATE TEXT

The **Label** Web server control is used for displaying text on a Web page. You can work with the **Label** control in your code and programmatically change the properties of the label, including the text that appears.

You can use the **Label** Web control to identify the purpose of controls that you arrange on a Web form. For example, you can have 3 **TextBox** controls on a form. You need to identify what each **TextBox** contains.

The **Label** Web control will reside on a server form on your ASP.NET page. The **Label** control is created

with the `<ASP: Label>` tag. You will need an **ID** attribute to give the control a name, which is how you reference it in code. As with other Web server controls, set the **RUNAT** attribute to **Server**. Like the **Image** Web server control, the **Label** Web server control does not support any events.

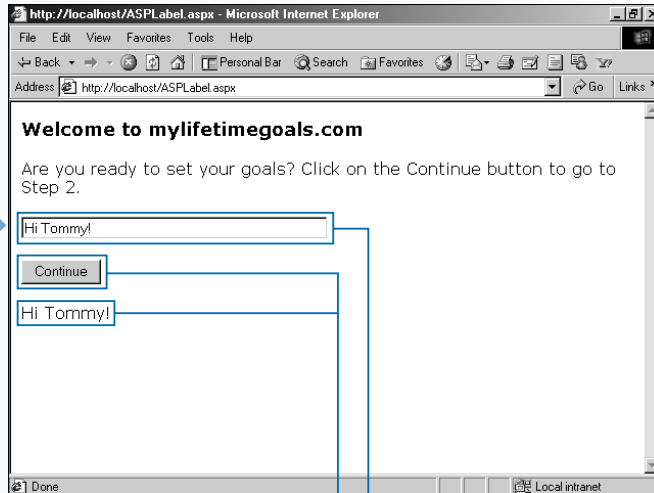
For example, you can have a form that takes input using a **TextBox** control. After filling in the text box, users can click a button, which fires off an event that can read what was in the text box and update a **Label** control on the form with this value.

## MANIPULATE TEXT

```

Untitled - Notepad
File Edit Format Help
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Button_OnClick(object Source, EventArgs e) {
    labelMessage.Text = textboxMessage.Text;
}
</SCRIPT>
</HEAD>
<BODY>
<FONT face="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
Are you ready to set your goals? Click on the Continue button to go to Step 2.

<FORM RUNAT="Server">
<P/>
<ASP:TEXTBOX ID="textboxMessage" TEXT="Type your welcome message here."
RUNAT="Server" WIDTH="300px"/>
<P/>
<ASP:BUTTON ID="buttonContinue" RUNAT="Server" onClick="Button_OnClick"
TEXT="Continue"/>
<P/>
<ASP:LABEL ID="labelMessage" RUNAT="Server" />
</FORM>
</FONT>
  
```



**1** Open the `WebWelcomeTemplate.aspx` template from the Code Templates directory.

**2** Add a **Text** box to the page.

■ Note the **Label** control on the page.

**3** Add a button to the pages that calls the `Button_OnClick` event.

**4** Modify the code in the `Button_OnClick` event so that the **label** is updated with the text from the text box.

**5** Save the file and request it from the Web server.

*Note: See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.*

**6** Type a message in the text box.

**7** Click the Submit button.

■ A message appears.

# ADD A PLACEHOLDER FOR CONTROLS

**Y**ou can use the `Placeholder` Web server control to hold other controls on your ASP.NET Web pages. This is a convenient way to specify where on your page you want to put controls. It is especially useful if you want to create all of your form controls programmatically.

You can use the `Placeholder` Web server control almost anywhere on your ASP.NET Web page. It does not have to be on a server form, but if you are adding controls for a form, you should make sure that it is between the `<form>` tags. You should then create the

control and add the control to the placeholder using the `Placeholder.Controls.Add` method.

The `Placeholder` control does not generate any HTML back to the Web browser, but is used only to specify the location of controls that you add at runtime. `Placeholder` controls are very useful in dynamically loading controls on a Web page. If you do not know which control should be on the page or how many controls should be on the page until the page is requested, then you should use the `Placeholder` control.

## USE A PLACEHOLDER FOR CONTROLS

The screenshot shows two windows. On the left is a Notepad window titled 'Untitled - Notepad' containing the following code:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Page_Load(Object sender, EventArgs e) {
    HtmlAnchor htmlAnchorHome;
    htmlAnchorHome = new HtmlAnchor();
    htmlAnchorHome.InnerText = "Go to Home Page";
    htmlAnchorHome.HRef = "http://www.mylifetimegoals.com";
    placeholderHome.Controls.Add(htmlAnchorHome);
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
Be sure to visit the Home Page.
<FORM RUNAT="Server">
<ASP:PLACEHOLDER ID="placeholderHome" RUNAT="Server" />
</FORM>
```

On the right is a Microsoft Internet Explorer window titled 'http://localhost/ASPPlaceholder.aspx - Microsoft Internet Explorer'. The browser shows the rendered page with the following content:

```
Welcome to mylifetimegoals.com
Be sure to visit the Home Page.
Go to Home Page
```

**1** Open the `WebVisitHomepageTemplate.aspx` template from the Code Templates directory.

**2** Add a `Placeholder` control to the form.

**3** Create an `HTML` server control and set its properties.

**4** Add the control to the `Placeholder` control.

**5** Save the file and request it from the Web server.

*Note:* See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.

The `HtmlAnchor` control appears where the placeholder is located.

# PROVIDE A CONTAINER FOR CONTROLS

The `Panel` control can be used as a container for other controls on your ASP.NET Web pages. The control is very useful if you wish to hide a specific group of controls.

The `Panel` Web control resides on a server form on your ASP.NET page. The `Panel` control is created with the `<ASP: Panel>` tag. You will need an `ID` attribute to give the control a name, which is how you reference it in code. You can set the initial value of attributes when you declare the control. For example, you can set the `VISIBLE` attribute to `False`, which would initially hide the control.

You use the `Panel` control to break the Web page into sections of the Web page the user can view. Therefore, you treat the `Panel` control as a container for other controls. As the user interacts with your page, you can hide `Panels` or make them visible. This gives the effect of seeing multiple pages, when in fact they are viewing different `Panels` of the same page.

For example, you can create a page that first displays one panel. When the user finishes the page and clicks a `Submit` button, the first panel can be hidden and the second panel can be displayed. This would give users the impression that they have moved to another page.

## PROVIDE A CONTAINER FOR CONTROLS

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>

<FORM RUNAT="Server">

<ASP: PANEL ID="panelStep1" RUNAT="SERVER">
Are you ready to set your goals? Click on the Continue button to go to Step 2.
</ASP: PANEL>

</FORM>

</FONT>
</BODY>
</HTML>

```

**1** Open the `GenericTemplate.aspx` template from the Code Templates directory.

**2** Add a heading for the page.

**3** Create a server form.

**4** Create a `Panel` control and give it an ID.

**5** Type a message to the user.

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>

<FORM RUNAT="Server">

<ASP: PANEL ID="panelStep1" RUNAT="SERVER">
Are you ready to set your goals? Click on the Continue button to go to Step 2.
<P/>
<ASP: BUTTON ID="buttonContinue" RUNAT="Server" onClick="Button_OnClick"
TEXT="Continue"/>
</ASP: PANEL>

<ASP: PANEL ID="panelStep2" RUNAT="SERVER" VISIBLE="False">
You are on Step 2.
<P/>
</ASP: PANEL>

```

**6** Add a button for the user to click and call the `Button_OnClick` function for the `onClick` event.

**7** Create a `Panel` control and give it an ID and set the `VISIBLE` attribute for the control to `False`.

**8** Add a message to the user about being on the second step.



**Extra**

You can use the `Panel` control's properties to give an improved visual effect as the user moves to the next step in a process. For example, you can change the `BACKCOLOR` property to give a nice visual effect as the user moves from one panel to the next. You can also set the `HEIGHT` and the `WIDTH` properties for the panel.

**Example:**

```
<ASP: PANEL ID="panelStep1" HEIGHT="100px" WIDTH="300px"
  BACKCOLOR="Silver" RUNAT="SERVER" />
```

Are you ready to set your goals? Click the Continue button to go to Step 2.

```
<P />
```

```
<ASP: BUTTON ID="buttonContinue" RUNAT="Server"
  onClick="Button_OnClick" TEXT="Continue" />
```

```
</ASP: PANEL>
```

```
<ASP: PANEL ID="panelStep2" BACKCOLOR="Gold"
  HEIGHT="100px" WIDTH="300px" RUNAT="SERVER" VISIBLE="False" />
```

You are on Step 2.

```
<P />
```

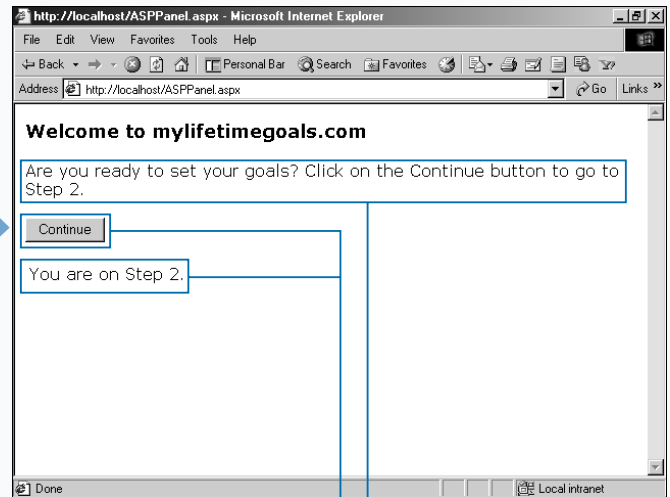
```
</ASP: PANEL>
```

```
Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Button_OnClick(object Source, EventArgs e) {
    panelStep1.Visible = false;
    panelStep2.Visible = true;
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
<FORM RUNAT="Server">
<ASP: PANEL ID="panelStep1" RUNAT="SERVER">
Are you ready to set your goals? Click on the Continue button to go to Step 2.
<P />
<ASP: BUTTON ID="buttonContinue" RUNAT="Server" onClick="Button_OnClick"
TEXT="Continue" />
</ASP: PANEL>
```

**9** Create the `Button_OnClick` function.

**10** Set the `Visible` property of the first panel to `False`.

**11** Set the `Visible` property of the second panel to `True`.



**12** Save the file and request it from the Web server.

*Note: See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.*

The first panel appears.

**13** Click the Continue button.

The second panel appears informing you that you are on step 2.

# DISPLAY ADVERTISEMENT BANNERS

Many Web sites use advertisement banners to generate income. ASP.NET gives you an `AdRotator` control that supports the process of displaying advertisements. The control supports displaying advertisements randomly. Each time a new page appears or is refreshed, an advertisement is selected.

The process of declaring the `AdRotator` control on your ASP.NET Web page is simple. Add the `<ASP:AdRotator RUNAT="Server">` tag to the place on the page where you want the banner to be displayed. Then, create or modify a separate XML format file, called the advertisement file. This file contains the details about the advertisements. The

details include the location of the graphic, the URL that the user is sent to when clicking the banner, and how often the advertisement should be displayed. Having this information in a separate XML file makes it easy to maintain the advertisements. The XML can be generated automatically through some server-side process or a programmer can modify it directly in the XML document.

For example, you can add advertisements to a Web page by declaring the `AdRotator` control. You can then modify an advertisement file with details on your ads. After you have done this, you can display an advertisement on your Web page. If you click Refresh, another advertisement may display.

## DISPLAY ADVERTISEMENT BANNERS

```

<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>

<FORM RUNAT="Server">
<ASP:ADROTATOR ID="adrotatorMyLifetimeGoals"
ADVERTISEMENTFILE="Ads.xml" BORDERWIDTH="1" RUNAT="Server" />
</FORM>

</FONT>
</BODY>
</HTML>

```

- 1 Open the `GenericTemplate.aspx` template from the Code Templates directory.
- 2 Add a heading for the page.

- 3 Add a server form to the page.
- 4 Add an `AdRotator` Web server control to the page and specify the location of the advertisement file.

```

<Advertisements>
<Ad>
<ImageUrl>images/mylifetimegoals1.gif</ImageUrl>
<NavigateUrl>http://www.mylifetimegoals.com</NavigateUrl>
<AlternateText>Go to mylifetimegoals.com</AlternateText>
<Keyword>Goalsetting</Keyword>
<Impressions>1</Impressions>
</Ad>
<Ad>
<ImageUrl>images/mylifetimegoals2.gif</ImageUrl>
<NavigateUrl>http://www.mylifetimegoals.com</NavigateUrl>
<AlternateText>Go to mylifetimegoals.com</AlternateText>
<Keyword>Goalsetting</Keyword>
<Impressions>1</Impressions>
</Ad>
<Ad>
<ImageUrl>images/mylifetimegoals3.gif</ImageUrl>
<NavigateUrl>http://www.mylifetimegoals.com</NavigateUrl>
<AlternateText>Go to mylifetimegoals.com</AlternateText>
<Keyword>Goalsetting</Keyword>
<Impressions>2</Impressions>
</Ad>
</Advertisements>

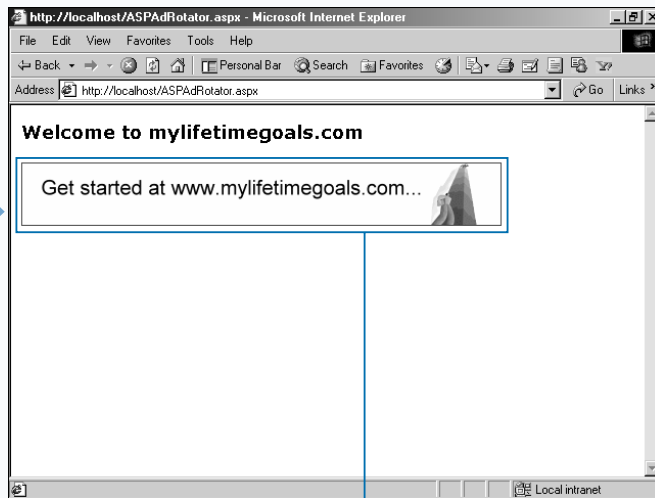
```

- 5 View the contents of the advertisement file.

**Extra**

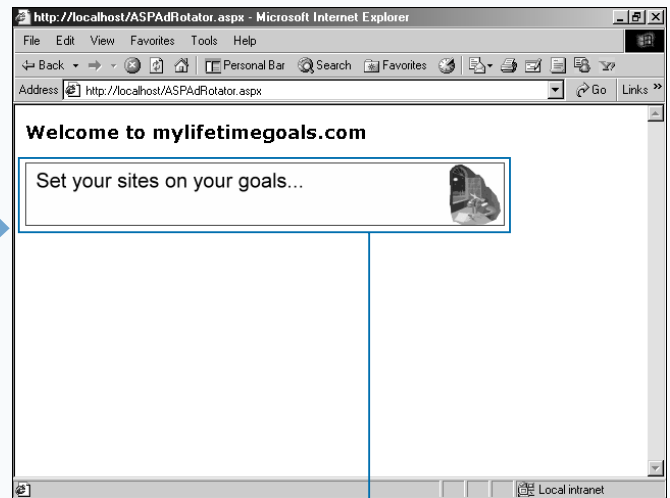
The elements of an Advertisement file are as follows:

- The root node of the XML file is the `<Advertisements>` element, which contains one too many `<Ad>` elements. Each of the `<Ad>` elements contain details on different advertisements. The child nodes to the `<Ad>` element are the `<ImageURL>`, `<NavigateURL>`, `<AlternateText>`, `<Keywords>`, and `<Impressions>` elements.
- The `<ImageURL>` element is the path and filename for the graphic to be displayed on the banner.
- The `<NavigateURL>` element is the URL that the user will be sent to when clicking the advertisement.
- The `<AlternateText>` element is what will be displayed if the Web browser has graphics turned off or displayed as help text when the user places the mouse pointer over the banner.
- The `<Keywords>` element describes the category under which the advertisement falls. You can use this element to filter out specific ads for different sections of your Web site.
- The control uses the `<Impressions>` element as a weighting for how often the advertisement should be displayed relative to the other advertisements.



**6** Save the file and request from the Web server.

■ An advertisement appears.



**7** Press F5 to refresh the Web page.

■ Another advertisement may appear.

*Note: See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.*

# VALIDATE REQUIRED FIELDS

You can use the `RequiredFieldValidator` control on your ASP.NET Web pages to specify which controls on your page require input. This is a convenient way to enable very basic validation on your pages. The `RequiredFieldValidator` checks to make sure that the user has changed a control's value from the initial value. You can use the control with most form controls, like the textbox and the dropdown list box.

To set this up, you have to create a `RequiredFieldValidator` control for each of the fields that you want to require input. You must then

declare the control at the location you want the error message to be displayed, and specify which control to validate by setting the `CONTROLTOVALIDATE` attribute equal to the `ID` of the control to validate. Finally, you specify the validation message for the user by setting the `TEXT` attribute.

For example, you can create a simple form that has the user input their login name. Because you always want the user to input something for the field, you can use the `RequiredFieldValidator` control to ensure that something was entered into the field.

## VALIDATE REQUIRED FIELDS

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
Please help us by taking the following survey.
<FORM RUNAT="Server">
</FORM>
</FONT>
</BODY>
</HTML>

```

- 1 Open the `GenericTemplate.aspx` template from the Code Templates directory.
- 2 Add a heading for the page.

- 3 Add a message to the user.
- 4 Add a server form to the page.

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
Please help us by taking the following survey.
<FORM RUNAT="Server">
What is your login name? <ASP:TEXTBOX ID="inputName" TEXTMODE="SingleLine"
TEXT="" WIDTH="200px" RUNAT="Server"/>
<P/>
<ASP:BUTTON TEXT="Submit" RUNAT="Server"/>
<ASP:REQUIREDFIELDVALIDATOR ID="requiredfieldvalidatorInputName"
RUNAT="Server"/>
</FORM>
</FONT>
</BODY>
</HTML>

```

- 5 Add another message to the user.
- 6 Add a `TextBox` control to the form.
- 7 Add a `Button` control to the form.

- 8 Add a `RequiredFieldValidator` control to the page and specify a name for the control by adding the `ID` attribute.

## Apply It

You can also validate other controls. For example, you can validate a drop-down list box. To do this, you need to set the `INITIALVALUE` attribute for the `REQUIREDFIELDVALIDATOR` element equal to the value that associates that the user has not selected an option.

### TYPE THIS:

```
<HTML>
<BODY>
<FONT FACE ="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
What suggestions do you have for improving our site?
<FORM RUNAT="Server">
<ASP:DROPDOWNLIST ID="dropdownlistSuggestions" RUNAT="Server">
  <ASP:LISTITEM>Choose</ASP:LISTITEM>
  <ASP:LISTITEM>Fewer Goals</ASP:LISTITEM>
  <ASP:LISTITEM>More Goals</ASP:LISTITEM>
  <ASP:LISTITEM>Same Number of Goals</ASP:LISTITEM>
</ASP:DROPDOWNLIST>
<P/><ASP:BUTTON TEXT="Submit" RUNAT="Server"/><P/>
<ASP:REQUIREDFIELDVALIDATOR ID="requiredfieldvalidatorInputName"
CONTROLTOVALIDATE="dropdownlistSuggestions" TEXT="You must select a
suggestion!" INITIALVALUE="Choose" RUNAT="Server"/>
</FORM>
</FONT>
</BODY>
</HTML>
```

### RESULT:

A page with a drop-down list appears that requires a selection to be made before submitting the page.

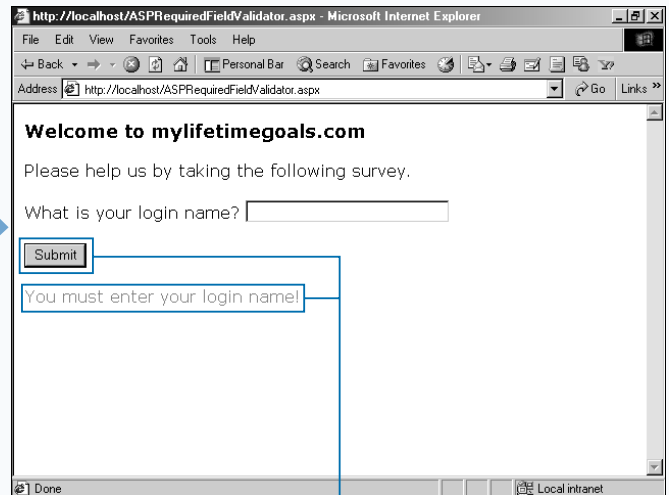
```
Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Please help us by taking the following survey.

<FORM RUNAT="Server">
What is your login name? <ASP:TEXTBOX ID="inputName" TEXTMODE="SingleLine"
TEXT="" WIDTH="200px" RUNAT="Server"/>
<P/>
<ASP:BUTTON TEXT="Submit" RUNAT="Server"/>
<P/>
<ASP:REQUIREDFIELDVALIDATOR ID="requiredfieldvalidatorInputName"
CONTROLTOVALIDATE="inputName" TEXT="You must enter your login name!"
RUNAT="Server"/>
</FORM>

</FONT>
</BODY>
</HTML>
```

**9** Within the `RequiredFieldValidator` control, set the validation message by specifying the `TEXT` attribute.



**10** Save the file and request it from the Web server.

*Note:* See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.

*Note:* Do not fill in the login name box.

**11** Click the Submit button. A validation message appears.

# COMPARE TWO FIELDS FOR VALIDATION

You can compare two values on a form when submitting a form to the server. To do so, you can use the `CompareValidator` control that comes with ASP.NET. You can do this comparison based on a number of operations, including: checking the controls to see if they are equal to each other, if one is greater than or equal to another, or if one is less than another.

To use this validation control, you must first create the two controls to use as criteria for the validation. You then need to create a `CompareValidator` control for each comparison you would like to make. The syntax for declaring a `CompareValidator` control is `<ASP:COMPAREVALIDATOR RUNAT="Server">`.

After you declare the control, you must specify the control to validate with the `CONTROLTOVALIDATE` attribute, the control to compare with the `CONTROLTOCOMPARE` attribute, and finally the operator with the `OPERATOR` attribute. After you have set these, you can then designate the validation message to be displayed with the `TEXT` attribute.

For example, you can create a form that requires one of the answers to be less than or equal to another one of the answers. If the input is invalid, you would then display a validation message.

## COMPARE TWO FIELDS FOR VALIDATION

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimagoals.com</H3>
Please help us by taking the following survey.

<FORM RUNAT="Server">
</FORM>

</FONT>
</BODY>
</HTML>

```

- 1 Open the `GenericTemplate.aspx` template from the Code Templates directory.
- 2 Add a heading for the page.

- 3 Add a message to the user.
- 4 Add a server form to the page.

```

Untitled - Notepad
File Edit Format Help
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimagoals.com</H3>
Please help us by taking the following survey.

<FORM RUNAT="Server">
What is the number of times you have visited goal-setting sites this month?
<ASP:TEXTBOX ID="inputTimesVisitedGoalSites" TEXTMODE="SingleLine" TEXT=""
WIDTH="50px" RUNAT="Server"/>
<BR/>
What is the number of times you have visited www.mylifetimagoals.com this month?
<ASP:TEXTBOX ID="inputTimesVisitedMyLifetimeGoals" TEXTMODE="SingleLine"
TEXT="" WIDTH="50px" RUNAT="Server"/>
<P/>
<ASP:BUTTON TEXT="Submit" RUNAT="Server"/>
<P/>
<ASP:COMPAREVALIDATOR ID="comparevalidatorInputName" RUNAT="Server"/>
</FORM>

</FONT>
</BODY>
</HTML>

```

- 5 Add a message to the user and `TextBox` control to the form.
- 6 Add another message to the user and `TextBox` control to the form.

- 7 Add a `Button` control to the form.
- 8 Add the `CompareValidator` control to the page.

## Apply It

You can programmatically check to see if the values for two controls are equal.

### TYPE THIS:

```
<HTML>
<BODY>
<FONT FACE ="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
Please help us by taking the following survey.
<FORM RUNAT="Server">
How many goal-setting sites have visited you this month?
<ASP:TEXTBOX ID="inputTimesVisitedGoalSites"
TEXTMODE="SingleLine" TEXT="" WIDTH="50px" RUNAT="Server"/>
<BR/>
How many visits to www.mylifetimegoals.com this month?
<ASP:TEXTBOX ID="inputTimesVisitedMyLifetimeGoals"
TEXTMODE="SingleLine" TEXT="" WIDTH="50px" RUNAT="Server"/>
<P/><ASP:BUTTON TEXT="Submit" RUNAT="Server"/></P/>
<ASP:COMPAREVALIDATOR ID="comparevalidatorInputName"
CONTROLTOVALIDATE="inputTimesVisitedMyLifetimeGoals"
CONTROLTOCOMPARE = "inputTimesVisitedGoalSites"
OPERATOR="Equal" TYPE="String"
TEXT="Answer 1 must equal to Answer 2." RUNAT="Server"/>
</FORM>
</FONT>
</BODY>
</HTML>
```

### RESULT:

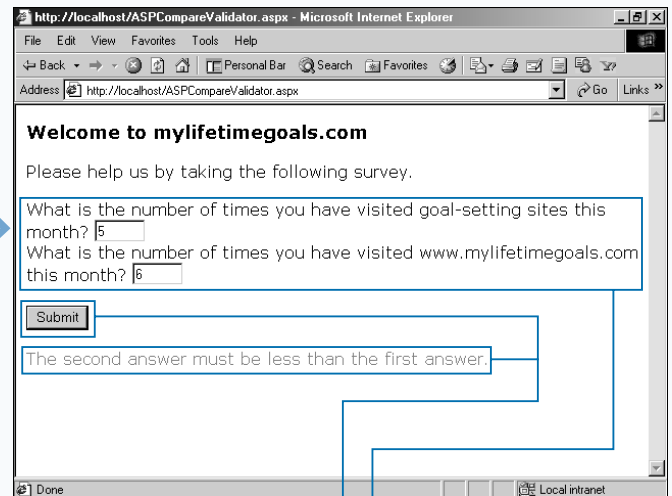
The result is a page that collects values for two questions. When the form is submitted, it checks to see if the values in the two text boxes are equal. If they are not equal, then a message is displayed.

```
Untitled - Notepad
File Edit Format Help
<H3>Welcome to mylifetimegoals.com</H3>
Please help us by taking the following survey.

<FORM RUNAT="Server">
What is the number of times you have visited goal-setting sites this month?
<ASP:TEXTBOX ID="inputTimesVisitedGoalSites" TEXTMODE="SingleLine" TEXT=""
WIDTH="50px" RUNAT="Server"/>
<BR/>
What is the number of times you have visited www.mylifetimegoals.com this month?
<ASP:TEXTBOX ID="inputTimesVisitedMyLifetimeGoals" TEXTMODE="SingleLine"
TEXT="" WIDTH="50px" RUNAT="Server"/>
<P/>
<ASP:BUTTON TEXT="Submit" RUNAT="Server"/>
<P/>
<ASP:COMPAREVALIDATOR ID="comparevalidatorInputName"
CONTROLTOVALIDATE="inputTimesVisitedMyLifetimeGoals"
CONTROLTOCOMPARE = "inputTimesVisitedGoalSites"
OPERATOR="LessThanEqual" TYPE="String" TEXT="The second answer must be less
than the first answer." RUNAT="Server"/>
</FORM>

</FONT>
</BODY>
</HTML>
```

**9** Within the `CompareValidator` control, set the validation message by specifying the `TEXT` attribute.



**10** Save the file and request it from the Web server.

*Note:* See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.

**11** Fill in the form with a smaller number for the first question.

**12** Click the Submit button.

A validation message appears.

# CHECK THE BOUNDARIES OF INPUT

Developers often need to validate a control based on a range of values. For these cases, you can use the `RangeValidator` Web server control. You can have different types of ranges, including ranges based on currency, dates, double-type data, integer-type data, and string data. If the user does not type input that falls within a valid range, a validation message appears, and the user gets a chance to correct the form and resubmit it.

After you place a control on a server form, you can then use the `<ASP:RANGEVALIDATOR RUNAT="Server">` tag to declare a

`RangeValidator` control. You must then specify the type of data you want to check, such as setting the type to `Date`. Then you need to specify the `MAXIMUMVALUE` and `MINIMUMVALUE` attributes to give the range for valid input. To set the validation message, you add the `TEXT` attribute and set it equal to the message.

For example, you can check a control to ensure that the user has put a date that falls within a specific range of dates. If the date falls outside this range, a validation message is displayed.

## CHECK THE BOUNDARIES OF INPUT

```

<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
Please help us by taking the following survey.
<FORM RUNAT="Server">
What is the date of the last time you visited www.mylifetimegoals.com?
</FORM>
</FONT>
</BODY>
</HTML>

```

**1** Open the `GenericTemplate.aspx` template from the Code Templates directory.

**2** Add a heading for the page.

**3** Add a message to the user.

**4** Add a server form to the page.

```

<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
Please help us by taking the following survey.
<FORM RUNAT="Server">
What is the date of the last time you visited www.mylifetimegoals.com?
<ASP:TEXTBOX
ID="InputLastDateVisitedMyLifetimeGoals" TEXTMODE="SingleLine" TEXT=""
WIDTH="100px" RUNAT="Server"/>
</P>
<ASP:BUTTON TEXT="Submit" RUNAT="Server"/>
</P>
<ASP:RANGEVALIDATOR ID="rangeValDate" RUNAT="Server"/>
</FORM>
</FONT>
</BODY>
</HTML>

```

**5** Add a message to the user and `TextBox` control to the form.

**6** Add a `Button` control to the form.

**7** Add the `RangeValidator` control to the page.



## Apply It

You can use the `Calendar` Web server control in combination with the `RangeValidator` control so the user does not need to type in a date, but can select the date from the `Calendar` control.

### TYPE THIS:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Date_Selected(object s, EventArgs e) {
    inputMessage.Text = calendarGoal.SelectedDate.ToShortDateString();
}
</SCRIPT></HEAD><BODY>
<FONT FACE ="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
Select the date you wish to accomplish this goal by.
<FORM RUNAT="Server">
<ASP:CALENDAR ID="calendarGoal"
onSelectionChanged="Date_Selected" RUNAT="Server" />
<P/><ASP:TEXTBOX ID="inputMessage" RUNAT="Server" />
<P/><ASP:BUTTON TEXT="SubmitDate" RUNAT="Server"/>
<P/><ASP:RANGEVALIDATOR ID="rangeValDate" TYPE="Date"
CONTROLTOVALIDATE="inputMessage" MAXIMUMVALUE="1/1/2002"
MINIMUMVALUE="1/1/1990"
TEXT="Please enter a date between 1/1/1990 and 1/1/2002."
RUNAT="Server"/>
</FORM>
</FONT>
</BODY>
</HTML>
```

### RESULT:

The result is a page that contains a calendar server control for selecting a date. When the date is selected and then submitted, the date is validated on the server. If the date does not fall within the specified date range, a message is displayed to the user.

```
Untitled - Notepad
File Edit Format Help

<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Please help us by taking the following survey.

<FORM RUNAT="Server">
What is the date of the last time you visited www.mylifetimegoals.com? <ASP:TEXTBOX
ID="input_LastDateVisitedMyLifetmeGoals" TEXTMODE="SingleLine" TEXT=""
WIDTH="100px" RUNAT="Server"/>
<P/>
<ASP:BUTTON TEXT="Submit" RUNAT="Server"/>
<P/>
<ASP:RANGEVALIDATOR ID="rangeValDate" TYPE="Date"
CONTROLTOVALIDATE="input_LastDateVisitedMyLifetmeGoals"
MAXIMUMVALUE="1/1/2002" MINIMUMVALUE="1/1/1990" TEXT="Please enter a
date between 1/1/1990 and 1/1/2002." RUNAT="Server"/>
</FORM>

</FONT>
</BODY>
```

**8** Within the `RangeValidator` control, set the validation message by specifying the `TEXT` attribute.

http://localhost/ASPRangeValidator.aspx - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://localhost/ASPRangeValidator.aspx

**Welcome to mylifetimegoals.com**

Please help us by taking the following survey.

What is the date of the last time you visited www.mylifetimegoals.com? 8/15/1972

Submit

Please enter a date between 1/1/1990 and 1/1/2002.

**9** Save the file and request it from the Web server.

*Note:* See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.

**10** Fill in the form with a date outside the range.

A validation message appears.

# VALIDATE WITH REGULAR EXPRESSIONS

You may encounter situations where you have to be very specific about validating the text that a user inputs. For these situations, you can use the `RegularExpressionValidator` control. This control lets you use regular expressions for your criteria to determine whether the input is valid. Regular expressions are a pattern-matching language used for processing text. For more information on pattern matching, see Microsoft's MSDN site ([msdn.microsoft.com](http://msdn.microsoft.com)).

Using the `RegularExpressionValidator` control is much like using the other validation controls. On the form containing the control you wish to validate, declare the control with the `<ASP:RegularExpressionValidator`

`RUNAT="Server">` tag. You then need to specify the control to validate with the `CONTROLTOVALIDATE` attribute. Next, use the `VALIDATIONEXPRESSION` attribute to indicate what you wish to use for the regular expression. Finally, specify the validation message using the `TEXT` attribute.

For example, you can use a simple regular expression to ensure that the user has filled in a text box with a ZIP Code. In the regular expression, you can just check to make sure that five numbers were entered. If the text box does not contain five numbers, you can display a message back to the users and enable them to correct the input and resubmit the form for validation and processing.

## VALIDATE WITH REGULAR EXPRESSIONS

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
Please help us by taking the following survey.

<FORM RUNAT="Server">
</FORM>

</FONT>
</BODY>
</HTML>

```

**1** Open the `GenericTemplate.aspx` template from the Code Templates directory.

**2** Add a heading for the page.

**3** Add a message to the user.

**4** Add a server form to the page.

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
Please help us by taking the following survey.

<FORM RUNAT="Server">
What is your 5-digit zip code? <ASP:TEXTBOX ID="inputZipCode"
TEXTMODE="SingleLine" TEXT="" WIDTH="50px" RUNAT="Server"/>
<P/>
<ASP:BUTTON TEXT="Submit" RUNAT="Server"/>
<P/>
<ASP:REGULAREXPRESSIONVALIDATOR
ID="regularexpressionvalidatorinputZipCode" RUNAT="SERVER" />
</FORM>

</FONT>
</BODY>
</HTML>

```

**5** Add a message to the user and `TextBox` control to the form.

**6** Add a `Button` control to the form.

**7** Add the `RegularExpressionValidator` control to the page.

**Extra**

Regular Expressions are commonly used for validating fields. Here are some useful examples of Regular Expressions.

CODE	DESCRIPTION
<code>\d{3}-\d{2}-\d{4}</code>	Testing for a valid Social Security Number.
<code>\d{5}(-\d{4})?</code>	Testing for valid US postal code.
<code>((\(\d{3}\) )?)   (\d{3}-)?)\d{3}-\d{4}</code>	Testing for a valid US phone number.
<code>((\(\d{3}\) )?)   (\d{3}-)?)\d{3}-\d{4}</code>	Testing for a valid e-mail address.
<code>http://([\w-]+\.)+[\w-](/[\w- ./?%=&amp;]*)?</code>	Testing for a valid URL.
<code>\d{4}-?\d{4}-?\d{4}-?\d{4}</code>	Testing for Visa Credit Card.

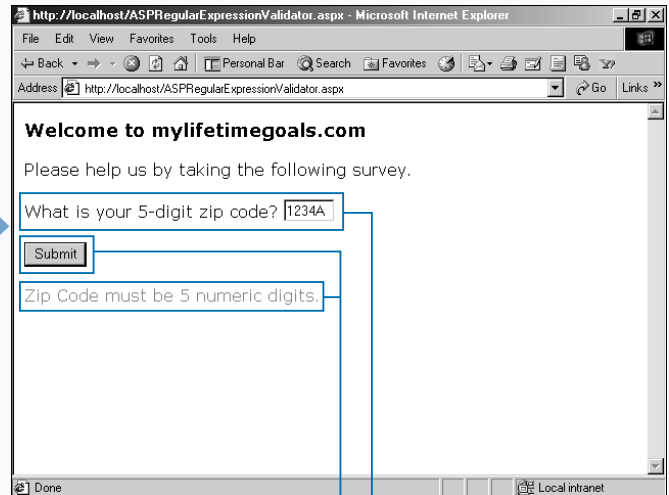
```

<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE = "Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Please help us by taking the following survey.

<FORM RUNAT="Server">
What is your 5-digit zip code? <ASP:TEXTBOX ID="inputZipCode"
TEXTMODE="SingleLine" TEXT="" WIDTH="50px" RUNAT="Server"/>
</P>
<ASP:BUTTON TEXT="Submit" RUNAT="Server"/>
</P>
<ASP:REGULAREXPRESSIONVALIDATOR
ID="regularexpressionvalidatorinputZipCode" RUNAT="SERVER"
CONTROLTOVALIDATE="inputZipCode" VALIDATIONEXPRESSION="\d{5}$"
DISPLAY="STATIC" TEXT="Zip Code must be 5 numeric digits."/>
</FORM>

</FONT>
    
```



**8** Within the `RegularExpressionValidator` control, set the validation message by specifying the `TEXT` attribute.

**9** Save the file and request it from the Web server.

*Note: See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.*

**10** Type in an invalid zip code.

**11** Click the Submit button.

A message appears.

# SUMMARIZE VALIDATION ERRORS

Most of your forms will perform validations on the different controls on your forms. To organize the validation errors that occur on your page, you can use the `ValidationSummary` Web server control to display a summary of the validation. This is especially convenient for users because they will see a list of all the validation errors for the form, not just the first validation error. Because the summary appears on the same page as the form, the user can address the issues immediately and resubmit the form.

You can use the `ValidationSummary` control to collect the validation results from all validated

controls. You should place the `ValidationSummary` control where you would like the summary message to be displayed. You can then specify the format for the summary. It is best to specify a header message for the validation summary, which is done with the `HEADERTEXT` attribute. The output of this control can be set with the `DISPLAYMODE` attribute.

For example, you can create a suggestion form that has all of the form controls and their associated validation controls. For this form, you can put a `ValidationSummary` control at the top to display all of the validation errors that were triggered upon submission.

## SUMMARIZE VALIDATION ERRORS

```

Untitled - Notepad
File Edit Format Help

<H3>Welcome to mylifetimegoals.com</H3>
Please help us by taking the following survey.

<FORM RUNAT="Server">
1. What is your Login name? <ASP:TEXTBOX ID="inputName"
TEXTMODE="SingleLine" TEXT="" WIDTH="200px" RUNAT="Server"/><BR>
2. What is the number of times you have visited goal-setting sites this month?
<ASP:TEXTBOX ID="inputTimesVisitedGoalSites" TEXTMODE="SingleLine" TEXT=""
WIDTH="50px" RUNAT="Server"/><BR>
3. What is the number of times you have visited www.mylifetimegoals.com this month?
<ASP:TEXTBOX ID="inputTimesVisitedMyLifetimeGoals" TEXTMODE="SingleLine"
TEXT="" WIDTH="50px" RUNAT="Server"/><BR>
4. What is the date of the last time you visited www.mylifetimegoals.com?
<ASP:TEXTBOX ID="inputLastDateVisitedMyLifetimeGoals" TEXTMODE="SingleLine"
TEXT="" WIDTH="100px" RUNAT="Server"/><BR>
5. What is your 5-digit zip code? <ASP:TEXTBOX ID="inputZipCode"
TEXTMODE="SingleLine" TEXT="" WIDTH="50px" RUNAT="Server"/>
<P>
<ASP:BUTTON TEXT="Submit" RUNAT="Server"/>
<P>
</FORM>
</FONT>

```

1 Open the `WebValidationTemplate.aspx` template from the Code Templates directory.

You can scroll down to view the form controls on the page.

```

Untitled - Notepad
File Edit Format Help

TEXTMODE="SingleLine" TEXT="" WIDTH="50px" RUNAT="Server"/>
<P>
<ASP:BUTTON TEXT="Submit" RUNAT="Server"/>
<P>
<ASP:REQUIREDFIELDVALIDATOR ID="requiredfieldvalidatorInputName"
CONTROLTOVALIDATE="inputName" ERRORMESSAGE="Please enter your login
name for Answer 1." DISPLAY="None" RUNAT="Server"/>
<ASP:COMPAREVALIDATOR ID="comparevalidatorInputName"
CONTROLTOVALIDATE="inputTimesVisitedMyLifetimeGoals"
CONTROLTOCOMPARE="inputTimesVisitedGoalSites"
OPERATOR="LessThanEqual" TYPE="String" ERRORMESSAGE="Please make sure
Answer 2 is less than Answer 3." DISPLAY="None" RUNAT="Server"/>
<ASP:RANGEVALIDATOR ID="rangeValDate" TYPE="Date"
CONTROLTOVALIDATE="inputLastDateVisitedMyLifetimeGoals"
MAXIMUMVALUE="1/1/2002" MINIMUMVALUE="1/1/1990" DISPLAY="None"
ERRORMESSAGE="Please enter a date between 1/1/1990 and 1/1/2002 for Answer
4." RUNAT="Server"/>
<ASP:REGULAREXPRESSIVATIONVALIDATOR
ID="regularexpressionvalidatorinputZipCode" RUNAT="SERVER"
CONTROLTOVALIDATE="inputZipCode" VALIDATIONEXPRESSION="\d{5}$"
DISPLAY="None" ERRORMESSAGE="Please enter a zip code with 5 numeric digits for
Answer 5." />
</FORM>

```

You can scroll down to view the validation controls on the page.

**Extra**

You can choose from three different formats when displaying your Validation Summary. They are specified using the `DISPLAYMODE` attribute for the `ValidationSummary` control.

The default mode is the bullet list.

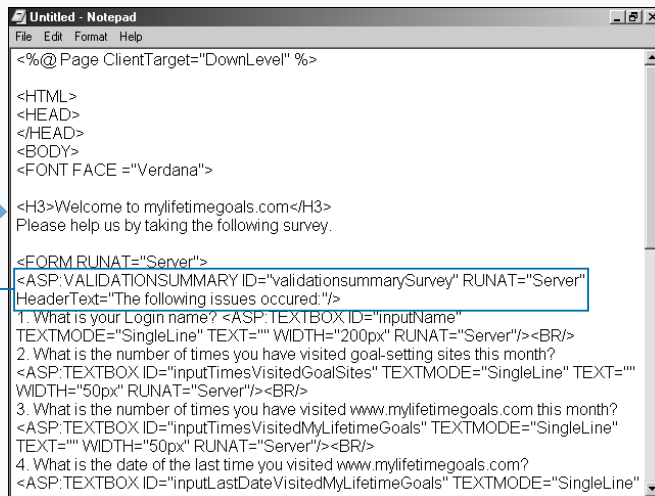
```
<ASP:VALIDATIONSUMMARY ID="validationsummarySurvey"
DISPLAYMODE="BulletList" RUNAT="Server"
HeaderText="The following issues occurred:" />
```

You can also specify a simple list.

```
<ASP:VALIDATIONSUMMARY ID="validationsummarySurvey"
DISPLAYMODE="List" RUNAT="Server" HeaderText="The
following issues occurred:" />
```

Finally, you can format the summary as a Paragraph.

```
<ASP:VALIDATIONSUMMARY ID="validationsummarySurvey"
DISPLAYMODE="SingleParagraph" RUNAT="Server"
HeaderText="The following issues occurred:" />
```



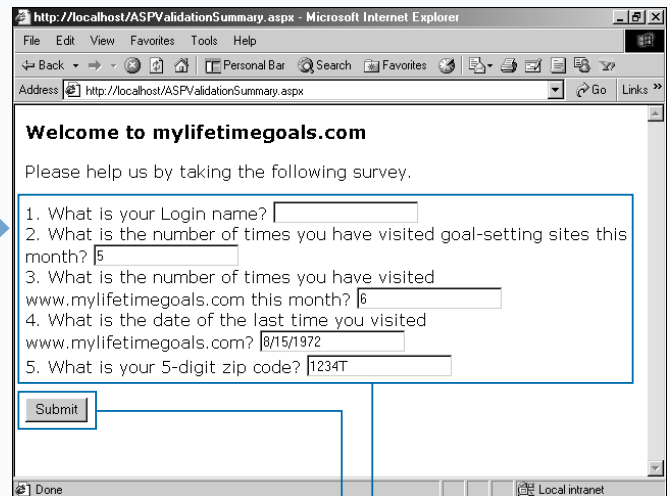
```
Untitled - Notepad
File Edit Format Help
<%@ Page ClientTarget="DownLevel" %>

<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Please help us by taking the following survey.

<FORM RUNAT="Server">
<ASP:VALIDATIONSUMMARY ID="validationsummarySurvey" RUNAT="Server"
HeaderText="The following issues occurred:" />
1. What is your Login name? <ASP:TEXTBOX ID="inputName"
TEXTMODE="SingleLine" TEXT="" WIDTH="200px" RUNAT="Server"/><BR/>
2. What is the number of times you have visited goal-setting sites this month?
<ASP:TEXTBOX ID="inputTimesVisitedGoalSites" TEXTMODE="SingleLine" TEXT=""
WIDTH="50px" RUNAT="Server"/><BR/>
3. What is the number of times you have visited www.mylifetimegoals.com this month?
<ASP:TEXTBOX ID="inputTimesVisitedMyLifetimeGoals" TEXTMODE="SingleLine"
TEXT="" WIDTH="50px" RUNAT="Server"/><BR/>
4. What is the date of the last time you visited www.mylifetimegoals.com?
<ASP:TEXTBOX ID="inputLastDateVisitedMyLifetimeGoals" TEXTMODE="SingleLine" />
```

**2** Add the `ValidationSummary` control to the page and set the properties for the control.



**3** Save the file and request it from the Web server.

*Note: See pages 20 to 25 for instructions on saving a file to the Web server and then requesting the file using the IIS Admin.*

**4** Fill out invalid input in each field.

**5** Click the Submit button.

A message appears showing all the validation errors.

# INTRODUCTION TO DATA ACCESS WITH ASP.NET

Data access is an integral part of creating dynamic web content. RDBMS (Relational Database Management System) storage is very commonly used in Web applications. ASP.NET provides data access to RDBMS storage via ADO.NET.

## BASICS OF DATA ACCESS

In this chapter, you will look at how to work with databases in ASP.NET using ADO.NET technology. The origin of where the data is provided is called the data source. To get to this data, you can use a couple of controls that are supplied by ADO.NET. These controls include a `Connection` object, which is used to make the connection to the database. An important property of the `Connection` object is the `ConnectionString`, which is used to specify how to open the database. Another control used for data access is the `Command` object, which is especially useful when executing a stored procedure. The `Command` object enables you to specify the parameters for the stored procedure. When you have a connection, you can then use a control called a `DataAdapter` to retrieve the data. You can use the `DataSet` object as the target for this data.

After you have the data from the data source, you will probably want to display this data on the Web page. To do this, you can bind the data to a control. There are several controls to which you can bind data. These include the `Repeater`, `DataList`, and `DataGrid` controls. The simplest control is the `Repeater` control. It enables you to create simple header, footer, and item templates for data. For more complex lists, you can work with the `DataList` control. This control allows for more complex formatting of your data. The `DataGrid` control is the most functional control available in ASP.NET for dealing with data. You have many formatting options for the `DataGrid` control, including paging and support for editable columns.

## CONTROLS USED TO ACCESS A DATABASE

### Connection

*Connections* are the starting point to data access. You need to give connection parameters to establish a connection. After those parameters are set (typically through the `ConnectionString` property), you will invoke the `Open` method to create an active connection. The following table provides Key Properties and Methods of the `SqlConnection` class, which is the class for connections with Microsoft SQL Server databases.

PROPERTY	DESCRIPTION
<code>ConnectionString</code>	(read/write) String used to open a SQL Server database.
<code>Database</code>	(read) Name of the current (or soon to be) connected database.
<code>DataSource</code>	(read) Name of SQL Server instance with which to connect.

## CONTROLS USED TO ACCESS A DATABASE (CONTINUED)

**Command**

The `Command` object in ADO.NET is very similar to its cousin ADO. Commands are important for stored procedures and you still want to use stored procedures in your data access routines (for both security and performance reasons). The following table provides Key Properties and Methods of the `SqlCommand` class, which is the class for commands with Microsoft SQL Server databases.

PROPERTY	DESCRIPTION
<code>CommandText</code>	(read/write) The T-SQL statement or stored procedure to execute at the data source.
<code>CommandType</code>	(read/write) A value indicating how the <code>CommandText</code> property is to be interpreted.
<code>Connection</code>	(read/write) The <code>SqlConnection</code> used by this instance of the <code>SqlCommand</code> .
<code>Parameters</code>	(read) The <code>SqlParameterCollection</code> .

METHOD	DESCRIPTION
<code>Cancel</code>	Cancels the execution of a <code>SqlCommand</code> .
<code>CreateParameter</code>	Creates a new instance of a <code>SqlParameter</code> object.
<code>ExecuteNonQuery</code>	Executes a T-SQL statement against the <code>Connection</code> and returns the number of rows affected.

**DataAdapter**

A `DataAdapter` object bridges the source data and the `DataSet` so that retrievals and updates can occur. The following table provides Key Properties and Methods of the `DataAdapter` class.

PROPERTY	DESCRIPTION
<code>AcceptChangesDuringFill</code>	(read/write) A value indicating whether <code>AcceptChanges</code> is called on a <code>DataRow</code> after it is added to the <code>DataTable</code> .
<code>TableMappings</code>	(read) A collection that provides the master mapping between a source table and a <code>DataTable</code> .

METHOD	DESCRIPTION
<code>Fill</code>	Adds or refreshes rows in the <code>DataSet</code> to match those in the data source using the <code>DataSet</code> name, and creates a <code>DataTable</code> named <code>Table</code> .
<code>GetFillParameters</code>	Retrieves the parameters set by the user when executing a SQL <code>SELECT</code> statement.
<code>Update</code>	Calls the respective <code>INSERT</code> , <code>UPDATE</code> , or <code>DELETE</code> statements for respective action in the specified <code>DataSet</code> from a <code>DataTable</code> named <code>Table</code> .

## CONTROLS USED TO DISPLAY DATA

**Repeater Control**

The simplest control for binding data is the `Repeater` control that enables you to create simple header, footer, and item templates for displaying data.

**DataList Control**

For more complex lists, you can work with the `DataList` control. Like the `Repeater` control, it uses templates for specifying how to display data.

**DataGrid Control**

The `DataGrid` control is the most functional control available in ASP.NET for dealing with data. You can define different types of columns with the `DataGrid`.

# DISPLAY REPEATING DATA

You can use the `Repeater` Web server control to format data into custom lists. To format the lists, you can use templates to define the layout of the data. This includes headers and footers, as well as alternating or separate rows of data.

You can work with the `Repeater` control by adding `<ASP:REPEATER RUNAT="Server">` to your ASP.NET Web page. You should give the control an `ID` so that you can reference it in code. To bind the control to the data source, you must first set the `DataSource` property and then call the `DataBind` method. You can then use the templates to format the output.

The `Repeater` control can support several different types of templates. Use an `ItemTemplate` when you want to format output for each row in the data source. The `ItemTemplate` is a required template. The `AlternatingItemTemplate` formats output as well, although the template is applied only to every other row of the data. You can use the `HeaderTemplate` and `FooterTemplate` for output that comes before the items and after the items, respectively. The `SeparatorTemplate` is used to specify formatting between rows of data. For example, you can specify a horizontal line in HTML.

## DISPLAY REPEATING DATA

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Here are some of the travel goals...

<FORM RUNAT="Server">
<P/>
<ASP:REPEATER ID="repeaterTable" RUNAT="Server">
</ASP:REPEATER>
</FORM>

</FONT>
</BODY>
</HTML>
  
```

**1** Open `GenericTemplate.aspx` from the Code Templates directory.

**2** Add a heading for the page.

**3** Add a message to the user.

**4** Add a Server Form to the page.

**5** Declare a `Repeater` control on the page and give it an `ID`.

```

Untitled - Notepad
File Edit Format Help
<FORM RUNAT="Server">
<P/>
<ASP:REPEATER ID="repeaterTable" RUNAT="Server">
  <HEADERTEMPLATE>
  <TABLE BORDER=1>
  <TR>
  <TD><B>Travel Goals</B></TD>
  </TR>
  </HEADERTEMPLATE>

  <ITEMTEMPLATE>
  <TR>
  <TD><%# Container.DataItem %></TD>
  </TR>
  </ITEMTEMPLATE>

  <FOOTERTEMPLATE>
  </TABLE>
  </FOOTERTEMPLATE>
</ASP:REPEATER>
</FORM>
  
```

**6** Add the header template tags and begin a new table with a heading.

**7** Add the item template tags and output the data items from the `Repeater` control.

**8** Add the footer template tags and end the table.



## Apply It

You can use the following `Repeater` control to display the data in comma delimited format by binding the `Reaper` control to an array list and using a comma as the `SeparatorTemplate`. The first set of code can be put into the `Page_Load` event and the second set into a server-side form to perform the databinding. See the full source code at `Chapter06/Code/ASPRepeater_ai.aspx`.

### TYPE THIS:

```
repeaterCommaDelimited.DataSource = alTravelGoals;
repeaterCommaDelimited.DataBind();
<ASP:REPEATER ID="repeaterCommaDelimited" RUNAT="Server">
<HEADERTEMPLATE>
<B>Travel Goals</B><BR/>
</HEADERTEMPLATE>
<ITEMTEMPLATE>
<%# Container.DataItem %>
</ITEMTEMPLATE>
<SEPARATORTEMPLATE>, </SEPARATORTEMPLATE>
</ASP:REPEATER>
```

### RESULT:

The items from the data source appear in a comma-separated list.

```

<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Page_Load(Object Sender, EventArgs E) {
    ArrayList alTravelGoals = new ArrayList();
    alTravelGoals.Add ("Travel to all seven continents");
    alTravelGoals.Add ("Travel to Asia, including Japan and China");
    alTravelGoals.Add ("Travel to Europe, including England, France, Spain, and
Italy");
    alTravelGoals.Add ("Travel to all 50 US states");
    alTravelGoals.Add ("Go to all the US National Parks");
    repeaterTable.DataSource = alTravelGoals;
    repeaterTable.DataBind();
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
Here are some of the travel goals...
<FORM RUNAT="Server">
```

http://localhost/ASPRepeater.aspx - Microsoft Internet Explorer

Address http://localhost/ASPRepeater.aspx

### Welcome to mylifetimegoals.com

Here are some of the travel goals...

Travel Goals
Travel to all seven continents
Travel to Asia, including Japan and China
Travel to Europe, including England, France, Spain, and Italy
Travel to all 50 US states
Go to all the US National Parks

**9** Add the `Page_Load` function to the page.

**10** Create an array list.

**11** Add some travel goals to the array list.

**12** Set the data source for the `Repeater` control to the array list.

**13** Bind the `Repeater` control to the array list data.

**14** Save the file and request it from the Web server.

A properly formatted table appears with the data from the array list.

# DISPLAY COMPLEX LISTS

You can use the `DataList` Web server control to format complex lists. This control shares many similar features with the `Repeater` control, but it has additional features, such as specifying the direction of the list as well as some additional template-formatting options.

You can work with the `DataList` control by adding `<ASP:DATALIST RUNAT="Server">` to your ASP.NET Web page. Give the control an `ID` attribute so that you can reference it in code. To bind the control to the data source, you should first set the `DataSource` property and then call the `DataBind` method. You can then use the templates to format the output.

The `DataList` supports a number of templates. For example, each row in the data source utilizes the `ItemTemplate`. The `AlternatingItemTemplate` formats every other row in the data source. If you want to have a table use different colors on alternating rows to aid in the readability of the table, you can use the `AlternatingItemTemplate`. You can use the `HeaderTemplate` and `FooterTemplate` to format a header and footer. You can specify what should be output between each row with the `SeparatorItem` template. `SelectedItemTemplate` describes the format when a user selects an item. `EditItemTemplate` describes the format when a user edits an item.

## DISPLAY COMPLEX LISTS

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Data" %>
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Here are some of the travel goals...

<FORM RUNAT="Server">
</FORM>
<ASP:DATALIST ID="datalistTravelGoals" RUNAT="Server">
<ITEMTEMPLATE>
<%# DataBinder Eval(Container.DataItem, "Goal") %>
</ITEMTEMPLATE>
</ASP:DATALIST>
</FORM>

</FONT>
</BODY>
</HTML>

```

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Data" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Page_Load(Object Sender, EventArgs E) {
    DataTable datatableGoals = new DataTable();
    DataRow datarowGoals;
    datatableGoals.Columns.Add(new DataColumn("Goal", typeof(string)));
    datatableGoals = datatableGoals.NewRow();
    datarowGoals[0] = "Travel to all seven continents";
    datatableGoals.Rows.Add(datarowGoals);
    datarowGoals = datatableGoals.NewRow();
    datarowGoals[0] = "Travel to Asia, including Japan and China";
    datatableGoals.Rows.Add(datarowGoals);
    datarowGoals = datatableGoals.NewRow();
    datarowGoals[0] = "Travel to Europe, including England, France, Spain, and
Italy";
    datatableGoals.Rows.Add(datarowGoals);
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">

```

**1** Open the `GenericTemplate.aspx` from the Code Templates directory.

**2** Add a heading and a message to the user.

**3** Add a Server Form to the page.

**4** Declare a `DataList` control on the page and give it an ID.

**5** Add an item template to the page and output the goal items.

**6** Import the `System.Data` namespace.

**7** Add the `Page_Load` function.

**8** Create a data table variable.

**9** Create a data row variable.

**10** Add a column to the data table.

**11** For each row, add a row and set the value for the Goal column.

## Apply It

You can select from many options when formatting the DataList. Type the following code into a server-side form. See the full source code at Chapter06/Code/ASPDataList\_ai.aspx.

### TYPE THIS:

```
<ASP:DATALIST ID="datalistTravelGoals" RUNAT="Server"
    BORDERCOLOR="Black"
    CELLPADDING="5"
    FONT-NAME="Verdana" FONT-SIZE="12px"
    HEADERSTYLE-FOREGCOLOR="White"
    HEADERSTYLE-FONT-BOLD="True"
    HEADERSTYLE-BACKCOLOR="Navy"
    ALTERNATINGITEMSTYLE-BACKCOLOR="LightBlue">
```

### RESULT:

This produces a formatted list of the data provided.

```
Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Data" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Page_Load(Object Sender, EventArgs E) {
    DataTable datatableGoals = new DataTable();
    DataRow datarowGoals;
    datatableGoals.Columns.Add(new DataColumn("Goal", typeof(string)));
    datarowGoals = datatableGoals.NewRow();
    datarowGoals[0] = "Travel to all seven continents";
    datatableGoals.Rows.Add(datarowGoals);
    datarowGoals = datatableGoals.NewRow();
    datarowGoals[0] = "Travel to Asia, including Japan and China";
    datatableGoals.Rows.Add(datarowGoals);
    datarowGoals = datatableGoals.NewRow();
    datarowGoals[0] = "Travel to Europe, including England, France, Spain, and
Italy";
    datatableGoals.Rows.Add(datarowGoals);
    DataView dataviewGoals = new DataView(datatableGoals);
    datalistTravelGoals.DataSource = dataviewGoals;
    datalistTravelGoals.DataBind();
}
</SCRIPT>
</HEAD>
```

**12** Create a `DataView` from the table that was created.

**13** Set the data source for the data list.

**14** Bind the data list to the data source.

http://localhost/ASPDataList.aspx - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://localhost/ASPDataList.aspx

**Welcome to mylifetimgoals.com**

Here are some of the travel goals...

- Travel to all seven continents
- Travel to Asia, including Japan and China
- Travel to Europe, including England, France, Spain, and Italy

Done Local intranet

**15** Save the file and request it from the Web server.

A properly formatted table appears with the data from the data table.

# DISPLAY SQL DATA

The DataGrid Web server control is a very flexible control for working with data. It supports advanced features to enable paging, editing of data, and sorting of data. The DataGrid control generates an HTML table (along with other HTML elements, depending on how the DataGrid is configured) to support these features. For server-side databinding, this is the most common control that you use. Without this control, you would be required to write many more lines of server side code to display data.

The process you use to work with a DataGrid resembles that of the Repeater and DataList controls if you just want to use the control for

displaying data. Because this example includes a database, you need to look at a few more new objects: `SqlConnection`, `SQLDataAdapter`, and `SQLDataSet`.

The `SqlConnection` object, found in the `System.Data.SqlClient` namespace, is used to create a connection from the Web server to the SQL server database. The `SQLDataAdapter`, also in the `System.Data.SqlClient` namespace, represents the connection and the commands to execute on the database. The `SQLDataSet`, from the `System.Data` namespace, will use the `SQLDataAdapter` to retrieve data from the SQL Server data source.

## DISPLAY SQL DATA

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Here are the are some books that will help you reach your career goals.
<P/>
<ASP:DATAGRID ID="datagridTitles" RUNAT="Server" SHOWHEADER="False"/>

</FONT>
</BODY>
</HTML>
  
```

**1** Open the `GenericTemplate.aspx` from the Code Templates directory.

**2** Add a heading for the page.

**3** Add a message to the user.

**4** Add the control to the page.

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
protected void Page_Load(Object sender, EventArgs e) {
    SqlConnection sqlconnectionPubs = new
    SqlConnection("server=(local)\NetSDK;uid=QSUser;pwd=QSPass;database=pubs");
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Here are the are some books that will help you reach your career goals.
<P/>
<ASP:DATAGRID ID="datagridTitles" RUNAT="Server" SHOWHEADER="False"/>

</FONT>
</BODY>
</HTML>
  
```

**5** Import the `System.Data` and `System.Data.SqlClient` namespaces.

**6** Add the `Page_Load` function.

**7** Create a `SqlConnection` object and use a connection string to connect to the database.

## Apply It

There are many options available to enhance the viewing and controlling of data with the `DataGrid`. To see some of these options, type the following code into a server-side form. See the full source code at `Chapter06/Code/ASPDataGrid_ai.aspx`.

### TYPE THIS:

```
<ASP:DATAGRID ID="datagridTitles" RUNAT="Server" SHOWHEADER="False"
  BORDERCOLOR="Black"
  CELLPADDING="5"
  FONT-NAME="Verdana" FONT-SIZE="12px"
  HEADERSTYLE-FORECOLOR="White"
  HEADERSTYLE-FONT-BOLD="True"
  HEADERSTYLE-BACKCOLOR="Navy"
  ALTERNATINGITEMSTYLE-BACKCOLOR="LightBlue"
/>
```

### RESULT:

This produces a formatted HTML table that contains the results of the query to the pubs database.

```
Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
protected void Page_Load(Object sender, EventArgs e) {
    SqlConnection sqlconnectionPubs = new
    SqlConnection("server=(local)\NetSDK;uid=QSUser;pwd=QSPassw;database=pub
s");

    SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter("select title,
notes, price from titles where type='business'", sqlconnectionPubs);

    DataSet datasetTitles = new DataSet();
    sqldataadapterTitles.Fill(datasetTitles, "titles");

    datagridTitles.DataSource=datasetTitles.Tables["titles"].DefaultView;
    datagridTitles.DataBind();
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
```

- 8 Create a `SQLDataAdapter` and set the `SQL` statement to retrieve business type books using the `SQLConnection` object.
- 9 Create a new `DataSet` object.

- 10 Add the `SQLDataAdapter` to populate the `DataSet`.
- 11 Set the `DataGrid` `Data Source` property to the `DataSet`.
- 12 Bind the `DataGrid` to the `DataSet`.

http://localhost/ASPDataGrid.aspx - Microsoft Internet Explorer

Address http://localhost/ASPDataGrid.aspx

Welcome to mylifetimegoals.com

Here are the are some books that will help you reach your career goals.

The Busy Executive's Database Guide	An overview of available database systems with emphasis on common business applications. Illustrated.	19.99
Cooking with Computers: Surreptitious Balance Sheets	Helpful hints on how to use your electronic resources to the best advantage.	11.95
You Can Combat Computer Stress!	The latest medical and psychological techniques for living with the electronic office. Easy-to-understand explanations.	2.99
Straight Talk About Computers	Annotated analysis of what computers can do for you: a no-hype guide for the critical user.	19.99

- 13 Save the file and request the file from the Web server.

■ A message appears.

# INSERT DATA INTO A SQL DATABASE

You can use ASP.NET to create Web pages that insert data into your SQL databases. To do this, you need to work with a couple of .NET framework objects. The first object that you need to use is the `Connection` object, which will be used to establish a connection to your database. The most important property for this object is `ConnectionString`, which can specify the server, the user ID and password, and the database to connect to when creating the connection.

When you have a connection, you can use the `Command` object to execute a SQL statement that

inserts a row of data into the database. You normally read this data from a form control on your ASP.NET Web page. After you set the SQL statement, you open the connection, execute the command, and then close the connection.

After you have inserted the data into the database, to confirm that the data insert was successful, you can use a bound `DataGrid` control. After the data is inserted, you will execute the `Command` object to select data that contains the new data (see page 132 for details on how to do this).

## INSERT DATA INTO A SQL DATABASE

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Here are the are some books that will help you reach your career goals.
<P/>
<ASP:DATAGRID ID="datagridTitles" RUNAT="Server" SHOWHEADER="False"/>

</FONT>
</BODY>
</HTML>
  
```

**1** Open the `GenericTemplate.aspx` from the Code Templates directory.

**2** Add a heading for the page.

**3** Add a message to the user.

**4** Add a `DataGrid` control to the page and set its properties.

**5** Import the `System.Data` and `System.Data.SqlClient` namespaces.

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
protected void Page_Load(Object sender, EventArgs e) {
    SqlConnection sqlconnectionPubs = new
    SqlConnection("server=(local)\NetSDK;uid=QSPUser;pwd=QSPPassword;database=pubs");
    SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter("select title,
    notes, price from titles where type='business'", sqlconnectionPubs);

    String insertCmd = "INSERT INTO titles(title_id, title, type, pub_id, price,
    advance, royalty, ytd_sales, notes, pubdate) VALUES('BU9999', 'How to Reach Your
    Business Goals', 'business', '0736', 25.00, 1000.00, 10, 1000, 'A practical how-to book
    on reaching even the most difficult business goals. Full of helpful tips, examples, and
    case studies.', '2001-06-12 00:00:00.000')";
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
  
```

**6** Create the `Page_Load` event.

**7** Create a `SqlConnection` object and use a connection string to connect to the database.

**8** Create a `SqlDataAdapter` object and set the SQL statement to retrieve business type books using the `SqlConnection` object.

**9** Create an insert command for the titles table and read it into a string variable.

**Apply  
It**

You are most likely inserting data based on what a user fills out on a form. To do this, you need to read this data from the form and put it in your INSERT SQL string. The following code shows how to read one of the parameters for an insert from a form control. This code executes in an event where the user clicks a Submit button. See Chapter06/Code/ASPInsert\_ai.aspx for the full source.

**TYPE THIS:**

```
String insertCmd = "INSERT INTO titles(title_id, title, type, pub_id, price, advance, royalty, ytd_sales, notes, pubdate) VALUES(@Id, 'How to Reach Your Business Goals', 'business', '0736', 25.00, 1000.00, 10, 1000, 'A practical how-to book on reaching even the most difficult business goals. Full of helpful tips, examples, and case studies.', '2001-06-12 00:00:00.000')";
SqlCommand sqlCommandTitles = new SqlCommand(insertCmd, sqlconnectionPubs);
sqlCommandTitles.Parameters.Add(new SqlParameter("@Id", SqlDbType.NVarChar, 6));
sqlCommandTitles.Parameters["@Id"].Value = inputTitleId.Text;
```

**RESULT:**

A page that asks for a book ID. When a valid ID is provided, you get a DataGrid control with the row that was added to the database.

```
notes, price from titles where type='business'", sqlconnectionPubs);

String insertCmd = "INSERT INTO titles(title_id, title, type, pub_id, price, advance, royalty, ytd_sales, notes, pubdate) VALUES('BU9999', 'How to Reach Your Business Goals', 'business', '0736', 25.00, 1000.00, 10, 1000, 'A practical how-to book on reaching even the most difficult business goals. Full of helpful tips, examples, and case studies.', '2001-06-12 00:00:00.000')";

SqlCommand sqlCommandTitles = new SqlCommand(insertCmd, sqlconnectionPubs);

sqlcommandTitles.Connection.Open();
sqlcommandTitles.ExecuteNonQuery();
sqlcommandTitles.Connection.Close();

DataSet datasetTitles = new DataSet();
sqldataadapterTitles.Fill(datasetTitles, "titles");

datagridTitles.DataSource=datasetTitles.Tables["titles"].DefaultView;
datagridTitles.DataBind();

}
</SCRIPT>
</HEAD>
```

**10** Use the insert command string and the connection object to create a `SqlCommand` object.

**11** Open, execute, and then close the connection to the database with the `SqlCommand` object.

**12** Populate the `DataSet` object.

**13** Set the `DataSource` and `DataBind` properties of the `DataGrid` on the page.

Here are the are some books that will help you reach your career goals.		
The Busy Executive's Database Guide	An overview of available database systems with emphasis on common business applications. Illustrated.	19.99
Cooking with Computers: Surreptitious Balance Sheets	Helpful hints on how to use your electronic resources to the best advantage.	11.95
You Can Combat Computer Stress!	The latest medical and psychological techniques for living with the electronic office. Easy-to-understand explanations.	2.99
Straight Talk About Computers	Annotated analysis of what computers can do for you: a no-hype guide for the critical user.	19.99
How to Reach Your Business Goals	A practical how-to book on reaching even the most difficult business goals. Full of helpful tips, examples, and case studies.	25

**14** Save the file and request it from the Web server.

A new record appears in the titles table.

# UPDATE DATA FROM A SQL DATABASE

You can use ASP.NET to create Web pages that can update data in a SQL database. Most of your applications require updating data that is persisted in a SQL Database. One way of updating data is by executing SQL UPDATE statements.

SQL UPDATE statements are typically built from information the user provides. The current data that is in the SQL Database is retrieved and displayed to the user. The user changes the values that need to be updated and then submits the information for updating. For example, you can update a price in a book database through a Web page.

To update data in a SQL database, you use the `SqlConnection` and `SqlCommand` objects. The

`SqlConnection` object creates a connection to the database. After you have a connection, you create an `SqlCommand` object and specify the SQL string to be executed against the database. Because you are most likely building this SQL string from user input, you can read the information off of an HTML or Web server control. After your SQL string is constructed, you can then open a connection using the `SqlConnection` object. To send your custom SQL statement to the database, you can use the `SqlCommand` object. After completing your database access code, make sure you close the connection to the database.

## UPDATE DATA FROM A SQL DATABASE

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Here are the are some books that will help you reach your career goals.
<P/>
<ASP:DATAGRID ID="datagridTitles" RUNAT="Server" SHOWHEADER="False"/>

</FONT>
</BODY>
</HTML>

```

**1** Open the `GenericTemplate.aspx` from the Code Templates directory.

**2** Add a heading for the page.

**3** Add a message to the user.

**4** Add a `DataGrid` control to the page and set its properties.

**5** Import the `System.Data` and `System.Data.SqlClient` namespaces.

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<HTML>
<HEAD>

<SCRIPT LANGUAGE="C#" RUNAT="Server">
protected void Page_Load(Object sender, EventArgs e) {
    SqlConnection sqlconnectionPubs = new
    SqlConnection("server=(local)\NetSDK;uid=QSUser;pwd=QSPass;database=pubs");
    SqlDataAdapter sqlDataAdapterTitles = new SqlDataAdapter("select title,
notes, price from titles where type='business'", sqlconnectionPubs);

    String updateCmd = "UPDATE titles SET price = 35.00 WHERE title_id =
'BU9999'";
</SCRIPT>
</HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Here are the are some books that will help you reach your career goals.
<P/>

```

**6** Create the `Page_Load` event.

**7** Create a `SqlConnection` object and use a connection string to connect to the database.

**8** Create a `SqlDataAdapter` and set the SQL statement to retrieve business type books using the `SqlConnection` object.

**9** Create an update command for the titles table and read it into a string variable.



## Apply It

You can update data based on what a user fills out on a form. To do this, you should read this data from the form and put it in your SQL UPDATE string. The following code shows how to read one of the parameters for an insert off a form control. This code executes in an event when the user clicks a Submit button. Please see Chapter 06/Code/ASPUpdate\_ai.aspx for the full source.

### TYPE THIS:

```
String updateCmd = "UPDATE titles SET price = 35.00
WHERE title_id = @Id";
SqlCommand sqlCommandTitles = new
SqlCommand(updateCmd, sqlconnectionPubs);
sqlcommandTitles.Parameters.Add(new
SqlParameter("@Id", SqlDbType.NVarChar, 6));
sqlcommandTitles.Parameters["@Id"].Value =
inputTitleId.Text;
```

### RESULT:

This produces a page that asks for a title ID. When a valid ID is provided, you get an updated price in a DataGrid control.

```
protected void Page_Load(Object sender, EventArgs e) {
    SqlConnection sqlconnectionPubs = new
    SqlConnection("server=(local)\NetSDK;uid=QSUser;pwd=QSPassWord;database=pub
    s");
    SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter("select title,
    notes, price from titles where type='business'", sqlconnectionPubs);
    String updateCmd = "UPDATE titles SET price = 35.00 WHERE title_id =
    'BU9999'";
    SqlCommand sqlCommandTitles = new SqlCommand(updateCmd,
    sqlconnectionPubs);
    sqlCommandTitles.Connection.Open();
    sqlCommandTitles.ExecuteNonQuery();
    sqlCommandTitles.Connection.Close();
    DataSet datasetTitles = new DataSet();
    sqldataadapterTitles.Fill(datasetTitles, "titles");
    datagridTitles.DataSource=datasetTitles.Tables["titles"].DefaultView,
    datagridTitles.DataBind();
}
</SCRIPT>
</HEAD>
```

**10** Use the insert command string and the connection object to create a SqlCommand object.

**11** Open, execute, and then close the connection to the database with the SqlCommand object.

**12** Populate the DataSet object.

**13** Set the DataSource and DataBind properties of the DataGrid on the page.

Here are the are some books that will help you reach your career goals.		
The Busy Executive's Database Guide	An overview of available database systems with emphasis on common business applications. Illustrated.	19.99
Cooking with Computers: Surreptitious Balance Sheets	Helpful hints on how to use your electronic resources to the best advantage.	11.95
You Can Combat Computer Stress!	The latest medical and psychological techniques for living with the electronic office. Easy-to-understand explanations.	2.99
Straight Talk About Computers	Annotated analysis of what computers can do for you: a no-hype guide for the critical user.	19.99
How to Reach Your Business Goals	A practical how-to book on reaching even the most difficult business goals. Full of helpful tips, examples, and case studies.	35

**14** Save the file and request it from the Web server.

A record is updated from the titles table. The price is now \$35, as opposed to \$25.

# DELETE DATA FROM A SQL DATABASE

You can use ASP.NET to create Web pages that can delete data from your SQL Databases. This is not as common as updating and inserting data, but is possible from a Web Page. The most common use of deleting data is in administrative applications that are used to maintain data in your application.

As with inserting and updating data, you use the `SqlConnection` and `SqlCommand` objects to delete data from your SQL database. You use the `SqlConnection` object to create a connection to the database. After you have a connection, you create a `SqlCommand` object and specify the SQL string to be executed against the database. Because you are most

likely building this SQL string from user input, you can read the information off an HTML or Web server control. After your SQL string is formatted and set, you can then open a connection using the `SqlConnection` object. Next, you can execute the command. Finally, you should close the connection to the database.

There are alternatives when it comes to deleting rows of data from a database. Some developers will add a flag to a database table that indicates if the data is active or not. This active flag can be used to archive data when performing maintenance on your database.

## DELETE DATA FROM A SQL DATABASE

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Here are the are some books that will help you reach your career goals.
</P>
<ASP:DATAGRID ID="datagridTitles" RUNAT="Server" SHOWHEADER="False"/>

</FONT>
</BODY>
</HTML>

```

- 1 Open the `GenericTemplate.aspx` from the Code Templates directory.
- 2 Add a heading for the page.
- 3 Add a message to the user.

- 4 Add a `DataGrid` control to the page and set its properties.
- 5 Import the `System.Data` and `System.Data.SqlClient` namespaces.

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
protected void Page_Load(Object sender, EventArgs e) {
    SqlConnection sqlconnectionPubs = new
        SqlConnection("server=(local)\NetSDK;uid=QUser;pwd=QSPass;database=pub
            s");
    SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter("select title,
        notes, price from titles where type='business'", sqlconnectionPubs);
    String deleteCmd = "DELETE FROM titles WHERE title_id = 'BU9999'";
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Here are the are some books that will help you reach your career goals.
</P>
<ASP:DATAGRID ID="datagridTitles" RUNAT="Server" SHOWHEADER="False"/>

```

- 6 Create the `Page_Load` event.
- 7 Create a `SqlConnection` object and use a connection string to connect to the database.

- 8 Create a `SQLDataAdapter` object and set the `SQL` statement to retrieve business type books using the `SqlConnection` object.
- 9 Create a `delete` command for the titles table and read it into a string variable.

## Apply It

You probably delete data based on what a user fills out on a form. To do this, you need to read this data from the form and put it in your DELETE SQL string. The following code shows how to read one of the parameters for an insert off a form control. This code executes in an event where the user clicks a Submit button. Please see Chapter06/Code/ASPDelete\_ai.aspx for the full source.

### TYPE THIS:

```
String deleteCmd = "DELETE FROM titles WHERE title_id = @Id";
SqlCommand sqlCommandTitles = new SqlCommand(deleteCmd, sqlconnectionPubs);
sqlCommandTitles.Parameters.Add(new SqlParameter("@Id", SqlDbType.NVarChar, 6));
sqlCommandTitles.Parameters["@Id"].Value = inputTitleId.Text;
```

### RESULT:

This produces a page that asks for a title ID. When a valid ID is provided, you get an updated DataGrid control that shows a list of books that are in the pubs database (minus the record for the title that you deleted).

```

<SCRIPT LANGUAGE="C#" RUNAT="Server">
protected void Page_Load(Object sender, EventArgs e) {
    SqlConnection sqlconnectionPubs = new
    SqlConnection("server=(local)\NetSDK;uid=QSPUser;pwd=QSPassw;database=pubs");
    SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter("select title,
    notes, price from titles where type='business'", sqlconnectionPubs);

    String deleteCmd = "DELETE FROM titles WHERE title_id = 'BU9999'";

    SqlCommand sqlCommandTitles = new SqlCommand(deleteCmd,
    sqlconnectionPubs);

    sqlCommandTitles.Connection.Open();
    sqlCommandTitles.ExecuteNonQuery();
    sqlCommandTitles.Connection.Close();

    DataSet datasetTitles = new DataSet();
    sqldataadapterTitles.Fill(datasetTitles, "titles");

    datagridTitles.DataSource=datasetTitles.Tables["titles"].DefaultView;
    datagridTitles.DataBind();
}
</SCRIPT>

```

**10** Use the delete command string and the connection object to create a SqlCommand object.

**11** Open, execute, and then close the connection to the database with the SqlCommand object.

**12** Populate the DataSet object.

**13** Set the DataSource and DataBind properties of the DataGrid on the page.

Address: http://localhost/ASPDelete.aspx

### Welcome to mylifetimegoals.com

Here are the are some books that will help you reach your career goals.

The Busy Executive's Database Guide	An overview of available database systems with emphasis on common business applications. Illustrated.	19.99
Cooking with Computers: Surreptitious Balance Sheets	Helpful hints on how to use your electronic resources to the best advantage.	11.95
You Can Combat Computer Stress!	The latest medical and psychological techniques for living with the electronic office. Easy-to-understand explanations.	2.99
Straight Talk About Computers	Annotated analysis of what computers can do for you: a no-hype guide for the critical user.	19.99

**14** Save the file and request it from the Web server.

The record disappears from the titles table.

# SORT DATA FROM A SQL DATABASE

ASP.NET Web pages provide some nice sorting features that you can use when working with SQL Data. When you work with large sets of data, it is very important to have sorting capabilities.

First, you need to retrieve the desired data into a `DataView` using a `DataSet`. See page 132 to see how a `DataSet` is created. After you have this data in your `DataView`, you can use the `Sort` property to specify the column on which to sort. After you sort the data, you can set the data source for the control to the sort data and bind the data.

The `DataGrid` control provides the ability to sort data by clicking on column headers. This is a nice feature, and can easily be done by creating an event procedure for the `DataGrid`'s `OnSortCommand` event. In this event, you capture the `SortExpression` on the `DataGrid` control and pass it to a function that would sort the data and rebind to the `DataGrid` control. You can control the direction of the sort by appending `ASC` (for ascending order) and `DESC` (for descending order) to the end of the `Sort` property of the `DataView` that you use for binding to the `DataGrid` control.

## SORT DATA FROM A SQL DATABASE

```

Untitled - Notepad
File Edit Format Help
F%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Here are the are some books that will help you reach your career goals.
<P/>
<ASP:DATAGRID ID="datagridTitles" RUNAT="Server" SHOWHEADER="False"/>

</FONT>
</BODY>
</HTML>
  
```

**1** Open the `GenericTemplate.aspx` from the Code Templates directory.

**2** Add a heading for the page.

**3** Add a message to the user.

**4** Add a `DataGrid` control to the page and set its properties.

**5** Import the `System.Data` and `System.Data.SqlClient` namespaces.

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
protected void Page_Load(Object sender, EventArgs e) {
    SqlConnection sqlconnectionPubs = new
    SqlConnection("server=(local)\NetSDK;uid=QUser;pwd=QSPasssword;database=pubs");
    SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter("select title,
    notes, price from titles where type='business'", sqlconnectionPubs);
    DataSet datasetTitles = new DataSet();
    sqldataadapterTitles.Fill(datasetTitles, "titles");
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Here are the are some books that will help you reach your career goals.
<P/>
  
```

**6** Create the `Page_Load` event.

**7** Create a `SqlConnection` object and use a connection string to connect to the database.

**8** Create a `SQLDataAdapter` and set the SQL statement to retrieve business-type books using the `SqlConnection` object.

**9** Create a `DataSet` object.

**10** Populate the `DataSet` object.

## Apply It

You can sort data by using the `SortExpression` Property on the `DataGrid` control. The first set of code goes into a server-side form and the second part goes into a server-side script block. Please see `Chapter06/Code/ASPSort_ai.aspx` for the full source.

### TYPE THIS:

```
<ASP:DATAGRID ID="datagridTitles" RUNAT="Server" ALLOWSORTING="True" OnSortCommand="datagridTitles_Sort"/>
protected void datagridTitles_Sort
(Object sender, DataGridSortCommandEventArgs e) {
    BindGrid(e.SortExpression);
}
public void BindGrid(String sortfield) {
    SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter("select title, notes, price"+
"from titles where type='business'", sqlconnectionPubs);
    DataSet datasetTitles = new DataSet();
    sqldataadapterTitles.Fill(datasetTitles, "titles");
    DataView Source = datasetTitles.Tables["titles"].DefaultView;
    Source.Sort = sortfield;
    datagridTitles.DataSource=Source;
    datagridTitles.DataBind();
}
```

### RESULT:

This produces a `DataGrid` control that sorts the table by the column that is clicked.

```
Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
protected void Page_Load(Object sender, EventArgs e) {
    SqlConnection sqlconnectionPubs = new
    SqlConnection("server=(local)\NetSDK;uid=QSUser;pwd=QSPass;database=pub
s");
    SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter("select title,
notes, price from titles where type='business'", sqlconnectionPubs);

    DataSet datasetTitles = new DataSet();
    sqldataadapterTitles.Fill(datasetTitles, "titles");

    DataView dataviewTitle = datasetTitles.Tables["titles"].DefaultView;
    dataviewTitle.Sort = "title";

    datagridTitles.DataSource=dataviewTitle;
    datagridTitles.DataBind();
}
</SCRIPT>
</HEAD>
```

**11** Create a `DataView` based on the `DataSet` that was filled.

**12** Sort the `DataView` on the title column with the `Sort` command.

**13** Set the `DataSource` and `DataBind` properties of the `DataGrid` on the page.

http://localhost/ASPSort.aspx - Microsoft Internet Explorer

Welcome to mylifetimegoals.com

Here are the are some books that will help you reach your career goals.

Cooking with Computers: Surreptitious Balance Sheets	Helpful hints on how to use your electronic resources to the best advantage.	11.95
Straight Talk About Computers	Annotated analysis of what computers can do for you: a no-hype guide for the critical user.	19.99
The Busy Executive's Database Guide	An overview of available database systems with emphasis on common business applications. Illustrated.	19.99
You Can Combat Computer Stress!	The latest medical and psychological techniques for living with the electronic office. Easy-to-understand explanations.	2.99

**14** Save the file and request it from the Web server.

The table is sorted based on the Title column.

# EXECUTE STORED PROCEDURES

Using *stored procedures* in your applications produces faster and more secure data access as compared to running SQL Statements directly against your database. Stored procedures are precompiled SQL statements that can be cached in memory on your database server. Stored procedures are a good way to control what types of queries you allow your users to execute. Stored procedures can be allowed or disallowed based on what type of user is accessing the system. Also, by using stored procedures, you will have all your SQL in one place and not distributed throughout your code. This makes maintenance much easier.

Stored procedures can have zero to many parameters. Parameters enable you to pass data to

the stored procedure. This data can be used by the stored procedure in a `SELECT` statement to filter data with the `WHERE` clause. This parameter data can also be used in `INSERT`, `UPDATE`, or `DELETE` statements to modify data in your database. Like the ad hoc queries, you are probably reading a majority of the data for the parameters for the stored procedures from controls on your Web page.

With stored procedures, you can also raise explicit errors if a problem occurs while executing the SQL that is in a stored procedure. It is good practice to inspect for these errors after the stored procedure returns control back to your server-side code.

## EXECUTE STORED PROCEDURES

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Here are the Social Security Numbers for the authors with a royalty percentage of 50%.
<P/>
<ASP:DATAGRID ID="datagridTitles" RUNAT="Server" SHOWHEADER="False"/>

</FONT>
</BODY>
</HTML>

```

**1** Open the `GenericTemplate.aspx` from the Code Templates directory.

**2** Add a heading for the page.

**3** Add a message to the user.

**4** Add a `DataGrid` control to the page and set its properties.

**5** Import the `System.Data` and `System.Data.SqlClient` namespaces.

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
protected void Page_Load(Object sender, EventArgs e) {
    SqlConnection sqlconnectionPubs = new
    SqlConnection("server=(local)\NetSDK;uid=QSUser;pwd=QSPasssword;database=pub
    s");
    SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter("byroyalty",
    sqlconnectionPubs);

    sqldataadapterTitles.SelectCommand.CommandType =
    CommandType.StoredProcedure;

    sqldataadapterTitles.SelectCommand.Parameters.Add(new
    SqlParameter("@Percentage", SqlDbType.Int, 4));
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

```

**6** Create the `Page_Load` event.

**7** Create a `SqlConnection` object and use a connection string to connect to the database.

**8** Create a `SQLDataAdapter` and set stored procedure using the `SqlConnection` object.

**9** Set the command type for the `SQLAdapter`.

**10** Add a parameter to the `Command` object.

## Apply It

You can also read the parameters for your stored procedures from a `form` control on your ASP.NET Web page. After users complete a form, they can click a Submit button, and the data can be read off the form at that time. The first section of code goes into the body of an HTML page and the second section of code goes into the click event of the button. Please see `Chapter06/Code/ASPStoredProcedure_ai.aspx` for the full source.

### TYPE THIS:

```
<FORM RUNAT="Server">
<ASP:DATAGRID ID="datagridTitles" RUNAT="Server" SHOWHEADER="False" VISIBLE="False" />
<P/>
Enter the royalty percentage
<P/>
<ASP:TEXTBOX ID="inputPercentage" TEXTMODE="SingleLine" TEXT="" WIDTH="200px" RUNAT="Server" /><BR/>
<ASP:BUTTON OnClick="SubmitBtn_Click" TEXT="Submit" RUNAT="Server" />
</FORM>
sqldataadapterTitles.SelectCommand.Parameters["@Percentage"].Value =
inputPercentage.Text;
```

### RESULT:

This produces a page that asks for a royalty percentage. When the percentage is submitted, a `DataGrid` control that contains a list of author IDs that meet the stored procedures criteria results.

```
protected void Page_Load(Object sender, EventArgs e) {
    SqlConnection sqlconnectionPubs = new
    SqlConnection("server=(local)\NetSDK;uid=QSUser;pwd=QSPasswrd;database=pub
    s");
    SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter("byroyalty",
    sqlconnectionPubs);

    sqldataadapterTitles.SelectCommand.CommandType =
    CommandType.StoredProcedure;

    sqldataadapterTitles.SelectCommand.Parameters.Add(new
    SqlParameter("@Percentage", SqlDbType.Int, 4));
    sqldataadapterTitles.SelectCommand.Parameters["@Percentage"].Value =
    50;

    DataSet datasetTitles = new DataSet();
    sqldataadapterTitles.Fill(datasetTitles, "titleauthor");

    datagridTitles.DataSource=datasetTitles.Tables["titleauthor"].DefaultView,
    datagridTitles.DataBind();
}
</SCRIPT>
</HEAD>
<BODY>
```

11 Set the value for the command.

12 Create a `DataSet` object.

13 Populate the `DataSet` object.

14 Set the `DataSource` and `DataBind` properties of the `DataGrid` on the page.

http://localhost/ASPStoredProcedure.aspx - Microsoft Internet Explorer

Address http://localhost/ASPStoredProcedure.aspx

**Welcome to mylifetimagoals.com**

Here are the Social Security Numbers for the authors with a royalty percentage of 50%.

427-17-2319
846-92-7186
899-46-2035
998-72-3567

15 Save the file and request it from the Web server.

The authors with a royalty percentage of 50% appear.



# WORK WITH MASTER-DETAIL RELATIONSHIPS

Web developers commonly work with the Master-Detail relationship when creating ASP.NET Web pages. The Master-Detail relationship usually appears as a column containing a list of items. When a user clicks an item in that column, another page with details about that particular item appears. A classic example of this is a customer order system, where you have a list of customer orders and for each order you have one or many items that make up an order.

To create a Master-Detail relationship, you work with two pages. The first page contains a `DataGrid` bound to data from a SQL database. You must define one of

the columns in the `DataGrid` as the column where the user can click by using the `<ASP:HYPERLINKCOLUMN>` tag. You then need to specify the field with the `DATANAVIGATEURLFIELD` attribute, the page to be linked to and how to format the URL with the `DATANAVIGATEURLFORMATSTRING` attribute, and the text for the column with the `TEXT` attribute.

After you finish the master page, you have to create the detail page. On the detail page, you use the data that was passed via the URL to execute a SQL query against the database. You can then display the results with another `DataGrid` or any other bound control that best suits the detail data display.

## WORK WITH MASTER-DETAIL RELATIONSHIPS

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
protected void Page_Load(Object sender, EventArgs e) {
    SqlConnection sqlconnectionPubs = new
    SqlConnection("server=(local)\NetSDK;uid=QSUser;pwd=QSPasswor;database=pub
s");
    SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter("select title_id,
title, notes, price from titles where type='business'", sqlconnectionPubs);

    DataSet datasetTitles = new DataSet();
    sqldataadapterTitles.Fill(datasetTitles, "titles");

    datagridTitles.DataSource=datasetTitles.Tables["titles"].DefaultView,
    datagridTitles.DataBind();
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3>Welcome to mylifetimagoals.com</H3>
  
```

**1** Open `DataGridTemplate.aspx` from the Code Templates directory.

**2** Add the SQL statement to retrieve a couple of columns from the titles table.

```

Untitled - Notepad
File Edit Format Help
    datagridTitles.DataBind();
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3>Welcome to mylifetimagoals.com</H3>
Here are the are some books that will help you reach your career goals.
<P/>
<ASP:DATAGRID ID="datagridTitles" RUNAT="Server" SHOWHEADER="False">
<COLUMNS>
<ASP:HYPERLINKCOLUMN
    DATANAVIGATEURLFIELD="title_id"
    DATANAVIGATEURLFORMATSTRING="Detail.aspx?id={0}"
    TEXT="Details"
/>
</COLUMNS>
</ASP:DATAGRID>
</FONT>
</BODY>
</HTML>
  
```

**3** Add the `DataGrid` to the page and set properties.

**4** Configure a special column with the `<COLUMNS>` tag.

**5** Specify the `title_id` column as an `<ASP:HYPERLINKCOLUMN>` and set the other properties of the column.

**6** Save the file as the master file.



**Extra**

You can use the Hyperlink column to link an item on a master list to a detail table for the master item selected. To have a column represented by a button, you could also use the `ButtonColumn` or the `EditCommandColumn`. Here are the definitions for the `ButtonColumn` and `EditCommandColumn`.

**Example:**

```
<asp:ButtonColumn
  ButtonType="LinkButton|PushButton"
  Command="BubbleText"
  DataTextField="DataSourceField"
  DataTextFormatString="FormatString"
  FooterText="FooterText"
  HeaderImageUrl="url"
  HeaderText="HeaderText"
  ReadOnly="True|False"
  SortField="DataSourceFieldToSortBy"
  Text="ButtonCaption"
  Visible="True|False"
/>
```

```
<asp>EditCommandColumn
  ButtonType="LinkButton|PushButton"
  CancelText="CancelButtonCaption"
  EditText="EditButtonCaption"
  FooterText="FooterText"
  HeaderImageUrl="url"
  HeaderText="HeaderText"
  ReadOnly="True|False"
  SortField="DataSourceFieldToSortBy"
  UpdateText="UpdateButtonCaption"
  Visible="True|False"
/>
```

```
Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
protected void Page_Load(Object sender, EventArgs e)
{
    SqlConnection sqlconnectionPubs = new
    SqlConnection("server=(local)\NetSDK;uid=QSUser,pwd=QSPassword;database=pub
    s");
    string stringSelect = "select * from titles where title_id = '" +
    Request.QueryString["id"] + "'";
    SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter(stringSelect,
    sqlconnectionPubs);
    DataSet datasetTitles = new DataSet();
    sqldataadapterTitles.Fill(datasetTitles, "titles");
    datagridTitles.DataSource=datasetTitles.Tables["titles"].DefaultView,
    datagridTitles.DataBind();
}
</SCRIPT>
</HEAD>
<BODY>
```

**7** Open `DataGridTemplate.aspx` from the Code Templates directory.

**8** Add the SQL statement to retrieve the data based on what was passed through the URL.

**9** Save the file as the detail file.

http://localhost/ASPMasterDetail.aspx - Microsoft Internet Explorer

Address http://localhost/ASPMasterDetail.aspx

**Welcome to mylifetimegoals.com**

Here are the are some books that will help you reach your career goals.

	title_id	title	notes	price
<a href="#">Details</a>	BU1032	The Busy Executive's Database Guide	An overview of available database systems with emphasis on common business applications. Illustrated.	19.99
<a href="#">Details</a>	BU1111	Cooking with Computers: Surreptitious Balance Sheets	Helpful hints on how to use your electronic resources to the best advantage.	11.95
<a href="#">Details</a>	BU2075	You Can Combat Computer Stress!	The latest medical and psychological techniques for living with the electronic office. Easy-to-understand	2.99

http://localhost/ASPDetail.aspx?id=BU1032

**10** Request the master file from the Web server.

**11** Click the Details button. The full details about the title are displayed.

# WORK WITH XML DATA SOURCES

ASP.NET makes it easy for you to work with XML Data Sources. XML is a W3C ([www.w3c.org](http://www.w3c.org)) specified standard that is well accepted in the software development industry for describing data in text files. There are many companies that are adapting XML as a standard for transporting lightweight data. XML has become instrumental in having disparate systems to have a way to communicate to each other.

Sometimes you may wish to keep certain data in XML files on your Web server. XML is a very convenient way to store and transport data in your applications. You store XML documents in standard, nonbinary text

files. This makes it easy to work with the documents. You can use any text-based viewer to inspect your XML documents.

To work with the XML file, you must first work with the `FileStream` object to open the file. Next, you need a `StreamReader` object for reading the byte stream from the `FileStream` object. The `DataSet` object has a `ReadXML` method that you can use to read the stream. After it has been read in, you can use a `DataView` based on the `DataSet`. Finally, the `DataView` can be bound to a `DataGrid`.

## WORK WITH XML DATA SOURCES

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.IO" %>
<%@ Import Namespace="System.Data" %>
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Here are the goal categories and goal category Ids
<P/>
<ASP:DATAGRID ID="datagridTitles" RUNAT="Server" />

</FONT>
</BODY>
</HTML>

```

- 1 Open `GenericTemplate.aspx` from the Code Templates directory.
- 2 Add a heading for the page.
- 3 Add a message to the user.

- 4 Add a `DataGrid` control to the page.
- 5 Import the `System.IO` and the `System.Data` namespaces.

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.IO" %>
<%@ Import Namespace="System.Data" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
protected void Page_Load(Object sender, EventArgs e) {
    DataSet datasetGoals = new DataSet();
    FileStream filestreamGoals = new
FileStream(Server.MapPath("goals.xml"), FileMode.Open, FileAccess.Read);
    StreamReader streamreaderGoals = new StreamReader(filestreamGoals);
    datasetGoals.ReadXml(streamreaderGoals);
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Here are the goal categories and goal category Ids.
<P/>
<ASP:DATAGRID ID="datagridTitles" RUNAT="Server" />

</FONT>
</BODY>

```

- 6 Add the `Page_Load` function to the page.
- 7 Create a new `DataSet` object.
- 8 Create a `FileStream` object and open the XML file on the Web server in Read mode.

- 9 Create a `StreamReader` object and read in the file from the `FileStream` object.
- 10 Read the XML into the `DataSet`.
- 11 Close the `FileStream` object.

**Extra**

If you change the index on the DataSet, you get the list of all of the goals as opposed to the goal categories.

**Example:**

```
DataView Source = new
DataView(datasetGoals.Tables[1]);
```

ADO.NET provides disconnected data access by leveraging the simplicity and power of XML. The architecture of ADO.NET is very tightly bound to the .NET XML framework. ADO.NET and the .NET XML framework converge in the DataSet object. The native serialization format of the DataSet in XML is a perfect choice for moving data between tiers (including remote locations, like the client's browser).

XML is a key enabling technology for the .NET Platform. To create XML for your applications, you can select data out of Microsoft SQL Server 2000 as XML. Microsoft SQL Server 2000 has made enhancements to the OLE DB provider (SQLOLEDB) to allow XML documents to be set as command text and to return result sets as a stream.

```
Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.IO" %>
<%@ Import Namespace="System.Data" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
protected void Page_Load(Object sender, EventArgs e) {
    DataSet datasetGoals = new DataSet();

    FileStream filestreamGoals = new
    FileStream(Server.MapPath("goals.xml"), FileMode.Open, FileAccess.Read);
    StreamReader streamreaderGoals = new StreamReader(filestreamGoals);
    datasetGoals.ReadXml(streamreaderGoals);
    filestreamGoals.Close();
    DataView Source = new DataView(datasetGoals.Tables[0]);
    datagridTitles.DataSource = Source;
    datagridTitles.DataBind();
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Here are the goal categories and goal category Ids.
```

**12** Create a new DataView and initialize it with the values read into the dataset.

**13** Set the DataSource for the DataGrid.

**14** Bind the DataGrid.

http://localhost/ASPXMLData.aspx - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://localhost/ASPXMLData.aspx

**Welcome to mylifetimegoals.com**

Here are the goal categories and goal category Ids.

name	goal-category_Id
Travel	0
Career	1
Educational	2
Relationship	3
Health	4
Adventure	5
Spiritual	6
Financial	7

Done Local intranet

**15** Save the file and request it from the Web server.

The goal category names and IDs appear.

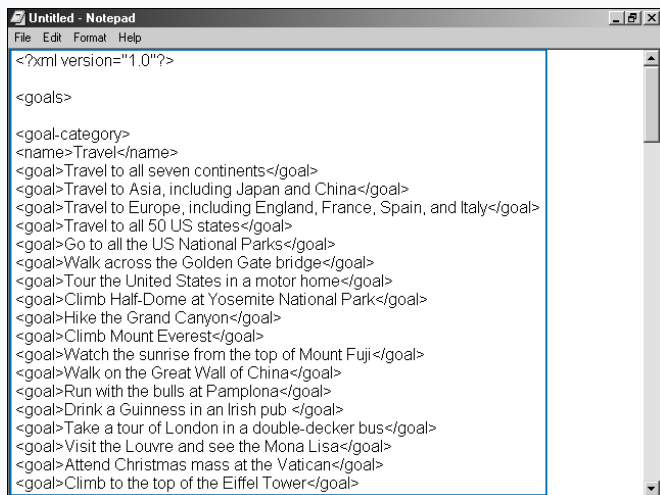
# TRANSFORM AND DISPLAY XML

A common task when working with XML documents is to transform an XML document with an XSL Transform document (also referred to as a XSLT style sheet or document). The XSLT document has information about how to format the data contained in the XML document. The XSLT document is described in XML like syntax. The specification for how to properly write an XSLT document can be found from the World Wide Web Consortium ([www.w3c.org](http://www.w3c.org)).

The purpose of doing transformations is to either create a new XML data source, format XML into a

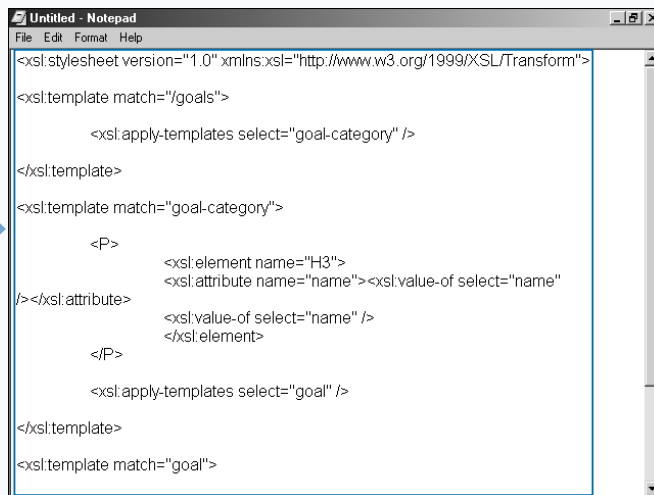
presentation markup language like HTML (HyperText Markup Language) or WML (Wireless Markup Language), or to do both. The `<ASP:XML>` makes performing transformations simple. To work with the `XML` control, you need to create a server form on an ASP.NET Web page. You must set two important attributes on the control. The first, `DOCUMENTSOURCE`, specifies the location of the XML document; the second, `TRANSFORMSOURCE`, indicates the location of the XSLT document. Add an `ID` attribute and give the control a unique name so that you can work with the control in code.

## TRANSFORM AND DISPLAY XML



```
<?xml version="1.0"?>
<goals>
  <goal-category>
    <name>Travel</name>
    <goal>Travel to all seven continents</goal>
    <goal>Travel to Asia, including Japan and China</goal>
    <goal>Travel to Europe, including England, France, Spain, and Italy</goal>
    <goal>Travel to all 50 US states</goal>
    <goal>Go to all the US National Parks</goal>
    <goal>Walk across the Golden Gate bridge</goal>
    <goal>Tour the United States in a motor home</goal>
    <goal>Climb Half-Dome at Yosemite National Park</goal>
    <goal>Hike the Grand Canyon</goal>
    <goal>Climb Mount Everest</goal>
    <goal>Watch the sunrise from the top of Mount Fuji</goal>
    <goal>Walk on the Great Wall of China</goal>
    <goal>Run with the bulls at Pamplona</goal>
    <goal>Drink a Guinness in an Irish pub </goal>
    <goal>Take a tour of London in a double-decker bus</goal>
    <goal>Visit the Louvre and see the Mona Lisa</goal>
    <goal>Attend Christmas mass at the Vatican</goal>
    <goal>Climb to the top of the Eiffel Tower</goal>
  </goal-category>
</goals>
```

**1** Open and review the XML document that you want to transform.



```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="goals">
    <xsl:apply-templates select="goal-category" />
  </xsl:template>
  <xsl:template match="goal-category">
    <P>
      <xsl:element name="H3">
        <xsl:attribute name="name"><xsl:value-of select="name" />
      </xsl:attribute>
      <xsl:value-of select="name" />
    </P>
    <xsl:apply-templates select="goal" />
  </xsl:template>
</xsl:stylesheet>
```

**2** Open and review the style sheet that is going to be used for the transformation.

## Apply It

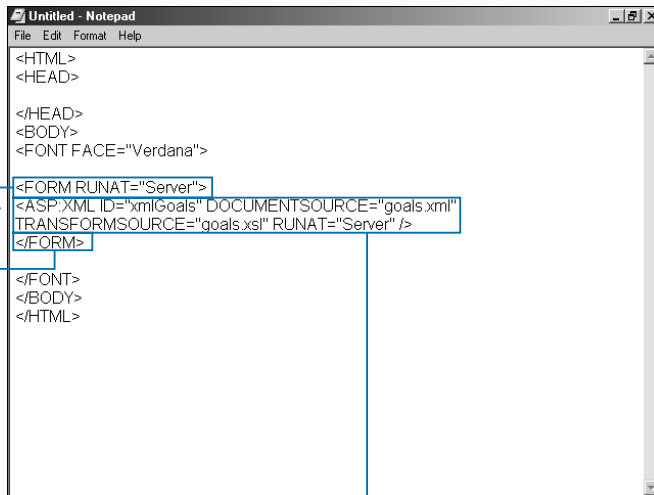
You can do transformations programmatically. This gives you a chance to ensure that the transformation executes without errors. The following code transforms an XML document using a style sheet.

### TYPE THIS:

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Xml" %>
<%@ Import Namespace="System.Xml.Xsl" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Page_Load(Object sender, EventArgs e) {
    XmlDocument xmldocumentGoals = new XmlDocument();
    xmldocumentGoals.Load(Server.MapPath("goals.xml"));
    XslTransform xsltransformGoals = new XslTransform();
    xsltransformGoals.Load(Server.MapPath("goals.xsl"));
    xmlGoals.Document = xmldocumentGoals;
    xmlGoals.Transform = xsltransformGoals;
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">
<FORM RUNAT="Server">
<ASP:XML ID="xmlGoals" RUNAT="Server" />
</FORM>
</FONT>
</BODY>
</HTML>
```

### RESULT:

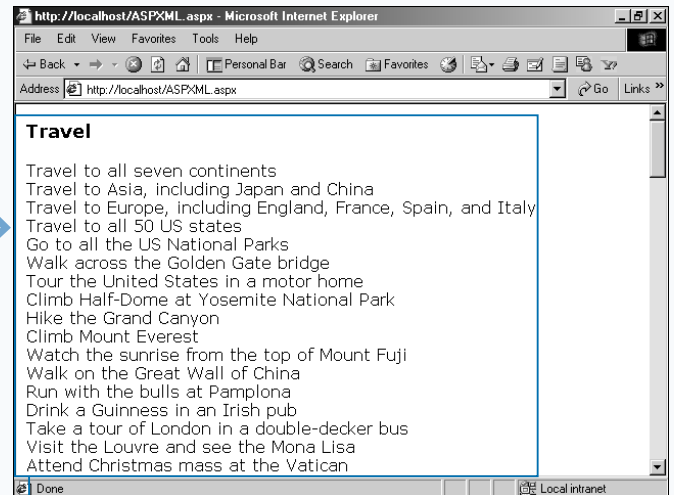
This produces an HTML page that is a result of transforming goals in an XML document to HTML.



**3** Open `GenericTemplate.aspx` from the Code Templates directory.

**4** Add a server form to the page.

**5** Add an `<ASP:XML>` tag and set its document source equal to the filename of the style sheet and the transform source equal to the XML document.



**6** Save the file and request the file from the Web server.

The XML document is transformed and output as HTML to the Web browser.

# INTRODUCTION TO WEB SERVICES

## WEB SERVICE BASICS

Web Services are units of application logic that provide data and services to other applications. Web Services are the next generation for programming Internet-based applications. Web Services combine the best aspects of component-based development and the Web. Applications can access Web Services through standard protocols and data formats like HTTP, XML, and SOAP.

ASP.NET Web Services provide the simplest way to implement Web Services. ASP.NET Web Services automatically generate Web Services Description Language, WSDL, and Web Services Discovery, Disco, files for your Web Services. You can use ASP.NET Web Services to implement a Web Service listener that

accesses a business façade implemented as a managed class using any compliant .NET language. The .NET Framework SDK also provides tools to generate proxy classes that client applications can use to access Web Services.

A Web Service interface is defined strictly in terms of the messages the Web Service accepts and generates. In ASP.NET, Web Services are implemented with Web Methods that have input parameters and a return value. Consumers of ASP.NET Web Services can be implemented on any platform in any programming language, as long as they can create and consume the messages defined for the Web Service interface.

## BENEFITS OF WEB SERVICES

Web Services enable you to expose business logic and data over the internet. The protocols for accessing Web Services are open standards which make them available for consumption or production by any platform.

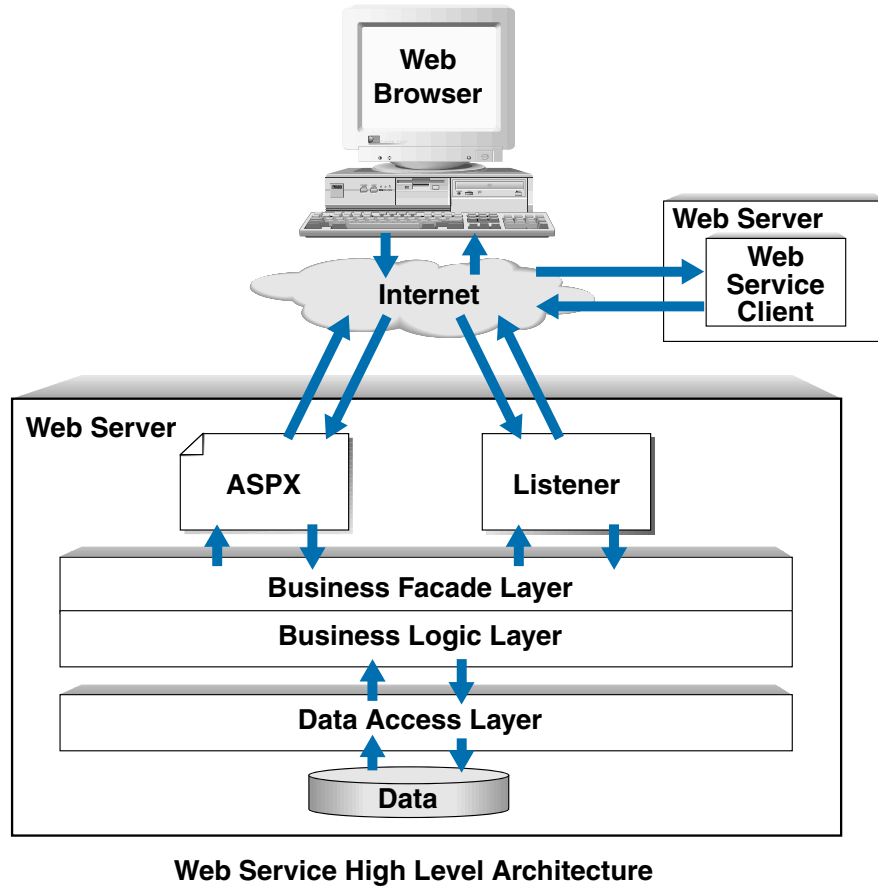
One of the large benefits of Web Services is the ability to pull data or apply business logic from many disparate systems and roll them into one application. For example, you can have a Web Service Client that accesses data from several servers' Web Services. This is advantageous for rolling up several data sources in your organization or for combining Web Services from different companies to create an application that leverages the "best in breed" in services that are provided on the Web.

As seen in the diagram, a user's browser can access a Web server's data in one of two ways. One way is the traditional way via an Active Server Page (ASPX) which

is a presentation layer that can connect to business services that obtain data from a SQL data store. The other way is by accessing Web Services through a Web Service Client that has the potential to access one to many Web Services on one to many Web servers. This provides new capabilities in distributed computing.

Another benefit to Web Services is security. Exposing Web Services over HTTP, where the communication port can be explicitly set, allows for better control over security. The security issues are easier to control than Distributed Component Object Model (DCOM) access, which is over the range of ports that are allocated for Remote Procedure Calls (RPC). Also, for HTTP access over a single port, there are many well-established products that can provide control over port access and HTTP commands in packets.

## BENEFITS OF WEB SERVICES (CONTINUED)



## WEB SERVICE PROTOCOLS AND STANDARDS

ASP.NET Web Services support service requests using SOAP over HTTP, as well as HTTP GET or POST.

The use of XML is central to the architecture of the .NET Platform, and Web Services is no exception. The most feature-rich access is through *Simple Object Access Protocol* (SOAP). SOAP is a lightweight XML protocol that defines the two way communication that occurs between Web Service Clients and Servers. For Microsoft-based Web Services, the SOAP specification defines a set of rules for how to use XML to represent data, define message envelopes — requests and responses — bindings to the HTTP protocol, and RPC over HTTP.

Although SOAP is the preferred way to access Web Services, you can also easily access a Service with an HTTP GET or POST. With the HTTP GET, you call a Web Service with parameters by providing a URL with a query string that holds the parameters. For example, you can call the Web Service with the following URL, `http://server/WebService.asmx/WebMethod?name=value`. Where `server` is the Web server path to the where the Web Service is located, `WebService.asmx` is the Web Services file, `WebMethod` is the desired Web Method on the Web Service, and `?name=value` is the parameter for the Web Method.

# CREATE A SIMPLE WEB SERVICE

**W**eb Services allow your business objects over HTTP. You can create a very simple Web Service in relatively few steps.

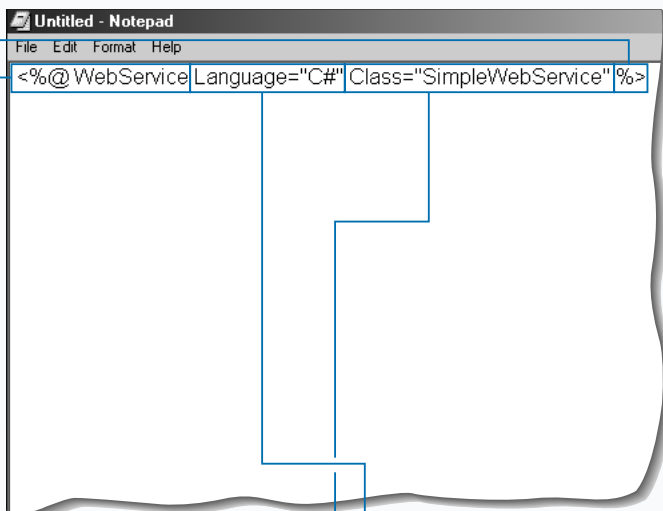
Web Services enable client applications to communicate to server components over the internet using the HTTP protocol. Web Services are requested over HTTP by HTTP-GET, HTTP-POST, or HTTP-SOAP. The most functional access method is HTTP-SOAP access. The SOAP access formats requests and responses from client to server with messages using XML for formatting the message contents.

To write a Web Service, you create a text file with an `.asmx` extension. You must add a directive at the top

of the page to specify the Web Service language, the class that implements the Web Service, and optionally the assembly containing the implementation. The assembly is required if you do not include the Web Service class inside of your ASMX file. The assembly needs to be in the `/bin` directory underneath the Web application that contains the Web Service.

For the class that is either embedded in the Web Service file, `*.asmx`, or in a separate file, `*.cs`, you must determine which methods in your class are exposed to Web Service clients. To expose methods in a class to Web Service clients, you must apply the `WebService` attribute to a public method.

## CREATE A SIMPLE WEB SERVICE



```

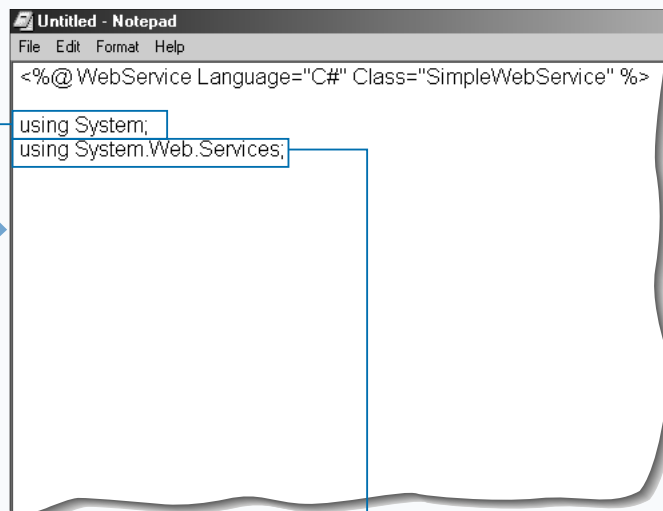
Untitled - Notepad
File Edit Format Help
<%@ WebService Language="C#" Class="SimpleWebService" %>
  
```

**1** Open a new document in your text editor.

**2** Add a `WebService` directive to the page.

**3** Set the Language in the directive.

**4** Give the `WebService` class a name.



```

Untitled - Notepad
File Edit Format Help
<%@ WebService Language="C#" Class="SimpleWebService" %>
using System;
using System.Web.Services;
  
```

**5** Create an alias for the `System` namespace.

**6** Create an alias for the `System.Web.Service` namespace.



**Apply It**

Using the `System.Random` class to create pseudo-random numbers and a simple switch statement you can create a Web Service that returns a random daily goal.

**TYPE THIS:**

```
<%@ WebService Language="C#" Class="SimpleWebService_ai" %>
using System;
using System.Web.Services;

public class SimpleWebService_ai : WebService {
    [WebMethod] public string GetRandomDailyGoal() {
        Random randomNumber = new Random();
        int intNumberOfGoals = 3;
        string stringGoal = "";
        int intRandomNumber = randomNumber.Next(intNumberOfGoals);
        switch(intRandomNumber){
            case 0:
                stringGoal = "Drink 8 glasses of water."; break;
            case 1:
                stringGoal = "Exercise for 30 minutes."; break;
            case 2:
                stringGoal = "Call your Mom."; break;
            default:
                goto case 2;
        }
        return stringGoal;
    }
}
```

**RESULT:**

A Web Service that will generate a random daily goal each time the service is requested.

```
Untitled - Notepad
File Edit Format Help
<%@ WebService Language="C#" Class="SimpleWebService" %>

using System;
using System.Web.Services;

public class SimpleWebService : WebService {
}
```

**7** Create the `SimpleWebService` class as a `WebService`.

```
Untitled - Notepad
File Edit Format Help
<%@ WebService Language="C#" Class="SimpleWebService" %>

using System;
using System.Web.Services;

public class SimpleWebService : WebService {
    [WebMethod] public String SayWelcomeStatement() {
        return "Welcome to www.mylifetimegoals.com!";
    }
}
```

**8** Create a `WebMethod` by placing the attribute before the method declaration.

**9** Add a `public` method that returns a string variable.

**10** Set the return value for the function.

**11** Save the file to the Web server with an `.asmx` extension.

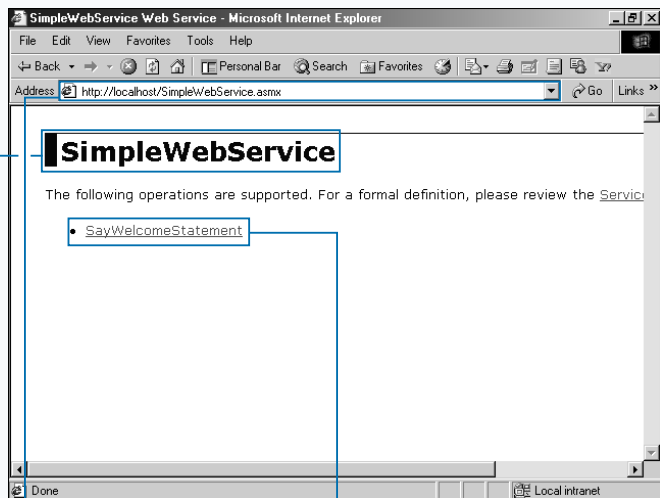
# TEST A WEB SERVICE

After you create your Web Service, you can test it. ASP.NET automatically generates pages for testing a Web Service when the Web Service ASMX file is requested with your Web browser. You can use these pages to see what is returned from the Web Service.

You do not need to create any additional pages for testing out a simple Web Service. Simply request the Web Service file — \*.asmx— with your Web browser. The first page that you see displays the Web Service class name and all of the Web methods that

are available for that Web Service class. You can click the Web Method name that you want to test. The next page displays an Invoke button that you can use to call the Web Method. If you have parameters for the Web Method, you will see a text box for entering each parameter value. After you click Invoke, another instance of Internet Explorer opens, and the XML that is generated from the request displays the results. If you do not have a Web Service to test, see page 152.

## TEST A WEB SERVICE



1 Open a Web browser and navigate to the Web Service file.

The Web Service class appears.

The WebMethod appears as a hyperlink.

2 Click the hyperlink.



3 Scroll down the page to view samples of a SOAP, HTTP Get, and HTTP Post request and response for the Web Service.

## Apply It

You can create a custom test page if you have special testing needs. The following is a Web page that you can use to create a custom testing page. The response from the Web Service will remain in the same Web browser window.

### TYPE THIS:

```
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

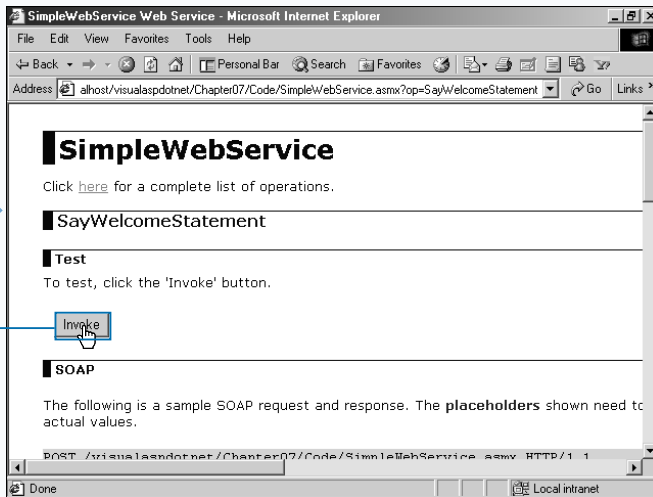
<H3>Test the Simple Web Service</H3>

<FORM ACTION='http://localhost/visualaspdotnet/Chapter07/Code/SimpleWebService.asmx/SayWelcomeStatement'>
<INPUT TYPE="Submit" VALUE="Test">
</FORM>

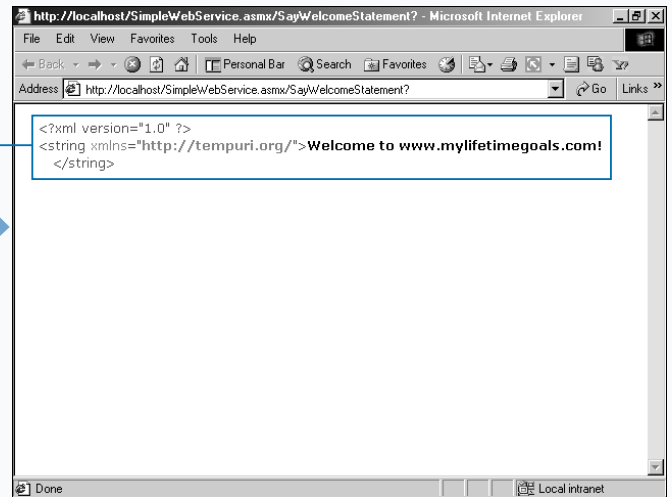
</FONT>
</BODY>
</HTML>
```

### RESULT:

A test page that is very similar to the automatically generated test page for a Web Service.



4 Click the Invoke button.



The HTTP Post response appears.

# USING A PARAMETER WITH A WEB SERVICE

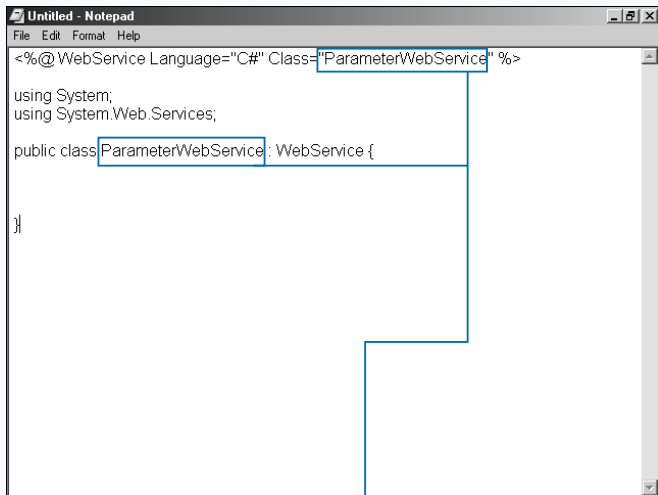
For most of your Web Services, you will want to pass parameters when invoking a Web Service method. These parameters give the method context to the Web client's specific needs from the Web Service. For example, a Web Service client can make a request for its goals for the next three months. Parameters that would be useful in this Web Service request are the customer's ID and the time frame for requesting goals (the next three months).

Creating a Web Service that accepts data for a parameter requires the same steps that it takes to create a Web Service without parameters. However,

in the function declaration for the `Web Method`, you must specify information on the parameter type and parameter name for each parameter needed to invoke the method. After you do this, you can use the data passed to the Web Service by referencing it by the parameter name.

Testing a Web Service method that has parameters is very similar to testing a method without parameters. The only difference is that the test page will include labeled text boxes for each parameter that is specified on the Web Service method.

## USING A PARAMETER WITH A WEB SERVICE



```

Untitled - Notepad
File Edit Format Help
<%@WebService Language="C#" Class="ParameterWebService" %>

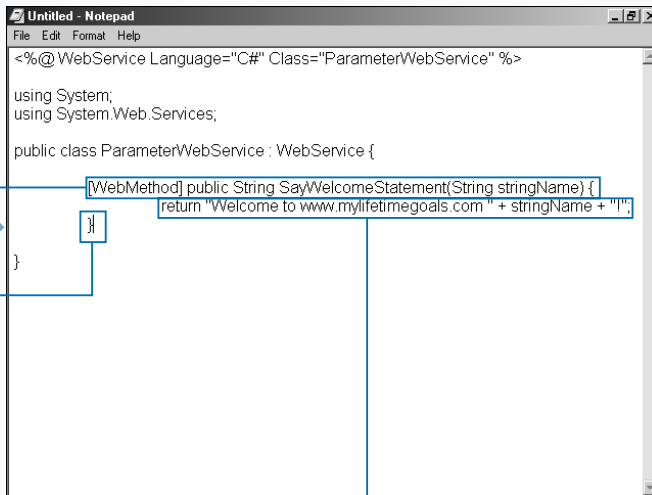
using System;
using System.Web.Services;

public class ParameterWebService : WebService {

}
  
```

**1** Open the `WebServicesTemplate.aspx` template from the Code Templates directory.

**2** Rename the class to `ParameterWebService`.



```

Untitled - Notepad
File Edit Format Help
<%@WebService Language="C#" Class="ParameterWebService" %>

using System;
using System.Web.Services;

public class ParameterWebService : WebService {

    [WebMethod] public String SayWelcomeStatement(String stringName) {
        return "Welcome to www.mylifetimegoals.com " + stringName + "!";
    }

}
  
```

**3** Create a `WebMethod` with a parameter in the function signature.

**4** Use the parameter the caller passes to the `WebMethod` for formatting the welcome statement.

## Apply It

You can create a Web Service method by leveraging other methods in the class that implement the Web Service. The following sample demonstrates this concept in a Web Service class (refer to this book's CD-ROM for an expanded example).

### TYPE THIS:

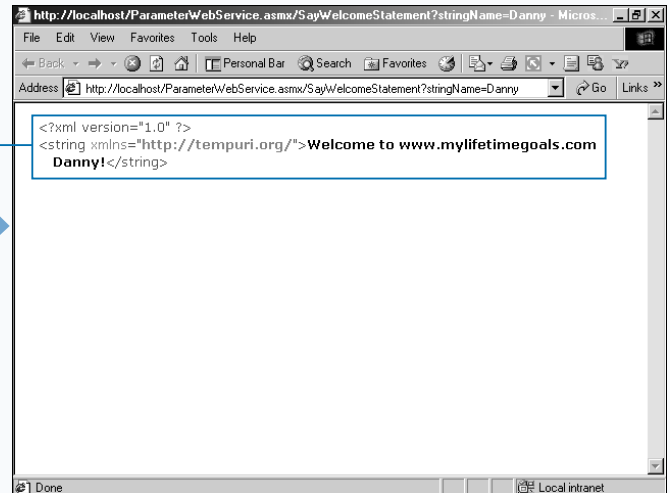
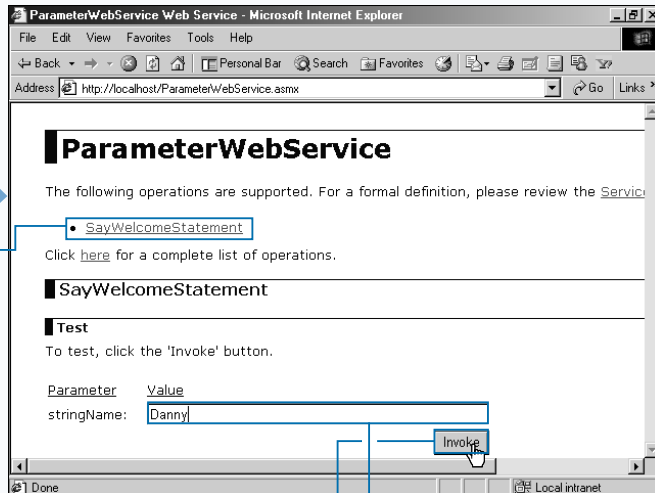
```
public class ParameterWebService_ai : Webservice {
    [WebMethod] public string GetRandomGoal(string stringGoalTimeLength) {
        stringGoalTimeLength = stringGoalTimeLength.ToUpper();
        if (stringGoalTimeLength != "DAILY" & stringGoalTimeLength != "YEARLY")
            stringGoalTimeLength = "DAILY";
        string stringGoal = "";
        int intRandomNumber = 1; // Replace this with random number.

        if (stringGoalTimeLength == "DAILY")
            stringGoal = DailyGoal(intRandomNumber);
        else if (stringGoalTimeLength == "YEARLY")
            stringGoal = YearlyGoal(intRandomNumber);

        return stringGoal;
    }
    public string YearlyGoal (int intRandomNumber) {
        string stringGoal = "Yearly Goals";
        return stringGoal;
    }
    public string DailyGoal (int intRandomNumber) {
        string stringGoal = "Daily Goal";
        return stringGoal;
    }
}
```

### RESULT:

A Web Service that will give a goal based on the time frame provided. This Web Service method will utilize other methods in the Web Service class.



5 Save the file and request it from the Web server.

6 Click the WebMethod to test.

7 Fill in your name for the parameter.

8 Click the Invoke button.

The parameterized welcome message appears.

# RETURN AN ARRAY FROM A WEB SERVICE

You can return more complex data types from your Web Services. For example, you can return arrays from a Web Service. Having the ability to return arrays from a Web Service enables you to pass back to the client a dynamic number of return values. With arrays, the return values will all be of the same data type, such as strings, integers, and so on.

To return an array from a Web Service, you need to first create a Web Service file containing a Web Method. The Web Method must define the array for the return type. In the Web Method, write the code that populates the array and specifies the return as the newly created array. When you test the Web Service, the array is designated by the

`ArrayOfDataType` element, and each member in the array will be a child element specified by the name of the data type. For example, the return that is an array of strings would generate the following XML:

```
<ArrayOfString>
  <string>First string member</string>
  <string>Second string member</string>
  <string>Third string member</string>
</ArrayOfString>
```

With this returned array, the client can work with the returned array to present the data in the client application. Note that arrays are zero based. Therefore, the first member of the array will be accessed with `arrayName[0]`.

## RETURN AN ARRAY FROM A WEB SERVICE

```
Untitled - Notepad
File Edit Format Help
<%@WebService Language="C#" Class="ArrayWebService" %>

using System;
using System.Web.Services;

public class ArrayWebService : WebService {

}
}
```

```
Untitled - Notepad
File Edit Format Help
<%@WebService Language="C#" Class="ArrayWebService" %>

using System;
using System.Web.Services;

public class ArrayWebService : WebService {

    [WebMethod]
    public string[] GetGoals() {
        string[] stringarrayGoals = new string[3];
    }

}
}
```

**1** Open the `WebServicesTemplate.asmx` template from the Code Templates directory.

**2** Rename the class to `ArrayWebService`.

**3** Create a `WebMethod` that returns a string array.

**4** Create a string array variable for holding the goals.

## Apply It

You can create a Web Service that returns a string array with a variable number of members based on an input parameter. For example, the following Web Service creates a string array with either one, two, or three goals returned based on the number of goals requested.

### TYPE THIS:

```
<%@ WebService Language="C#" Class="ArrayWebService_ai" %>
using System;
using System.Web.Services;

public class ArrayWebService_ai : WebService {
    [WebMethod]
    public string[] GetGoals(int intNumberOfGoals) {
        if (intNumberOfGoals > 3) intNumberOfGoals = 3;
        if (intNumberOfGoals < 1) intNumberOfGoals = 1;
        string[] stringarrayGoals = new string[intNumberOfGoals];
        stringarrayGoals[0] = "Regular exercise at the gym (3 days a week)";

        if (intNumberOfGoals == 2 | intNumberOfGoals ==3)
            stringarrayGoals[1] = "A patient better driver";

        if (intNumberOfGoals == 3)
            stringarrayGoals[2] = "Keep in contact with old friends";
        return stringarrayGoals;
    }
}
```

### RESULT:

An array of goals. The number of goals returned is dependent on the number the client provides.

```
Untitled - Notepad
File Edit Format Help
<%@WebService Language="C#" Class="ArrayWebService" %>

using System;
using System.Web.Services;

public class ArrayWebService : WebService {

    [WebMethod]
    public string[] GetGoals() {
        string[] stringarrayGoals = new string[3];
        stringarrayGoals[0] = "Regular exercise at the gym (3 days a
week)";

        stringarrayGoals[1] = "A patient better driver";
        stringarrayGoals[2] = "Keep in contact with old friends";
        return stringarrayGoals;
    }
}
```

**5** Read the goals into the array.

*Note: The array is zero based.*

**6** Return the string array.

```
http://localhost/ArrayWebService.asmx/GetGoals? - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/ArrayWebService.asmx/GetGoals?
<?xml version="1.0" ?>
<ArrayOfString xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
xmlns="http://tempuri.org/">
<string xmlns="">Regular exercise at the gym (3 days a week)</string>
<string xmlns="">A patient better driver</string>
<string xmlns="">Keep in contact with old friends</string>
</ArrayOfString>
```

**7** Save the file and test the Web Service.

The Web Service returns an array.

*Note: The ArrayOfString element is used.*

*Note: Nested string elements are used for the string array.*

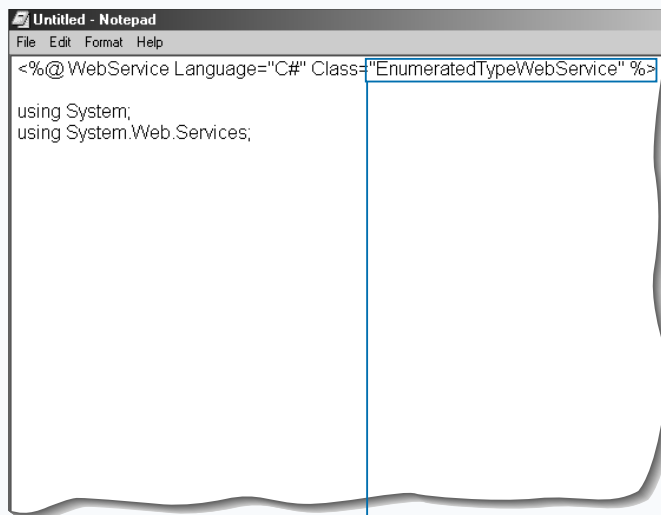
# RETURN AN ENUMERATED TYPE FROM A WEB SERVICE

Returning enumerations from a Web Service is useful when you have a set of fixed values for a variable. For example, you can define an enumeration to classify the types of goals a user can set in your application.

First you must add the definition of the enumeration into your Web Service file. After creating the enumeration, you can use the enumeration name as the return type for your Web Method. To return an enumeration, you can use the name of the enumeration and the desired member to return, separated by a period. For example, you can return the `Travel` member from the `GoalType` enumeration by using `return GoalType.Travel`.

To create an enumeration, you can set constant values for each enumeration or let the value for the constant be created automatically. For client applications that use Web Services with enumerations, you need to determine if the client application needs to work with the enumeration by its name or value. If you work with the enumeration by name, you need to define the enumeration in the client application to be able to access the value for the enumeration. If the client application is not aware of the enumeration, you will pass back the enumeration's value.

## RETURN AN ENUMERATED TYPE FROM A WEB SERVICE



```

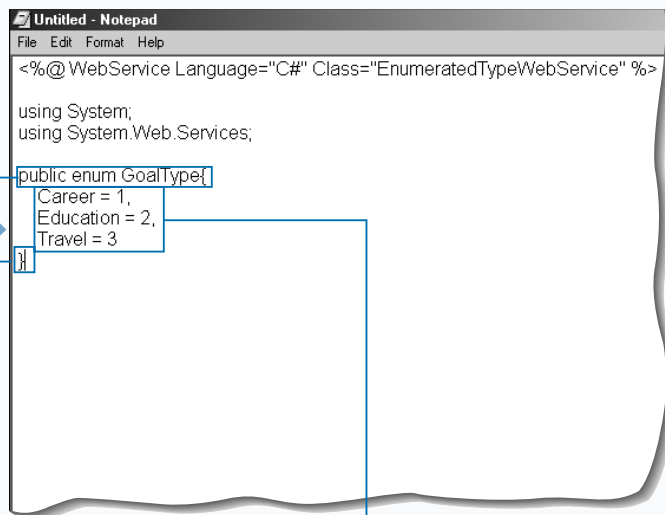
<%@ WebService Language="C#" Class="EnumeratedTypeWebService" %>

using System;
using System.Web.Services;

```

**1** Open the `WebServicesTemplate.aspx` template from the Code Templates directory.

**2** Rename the class to `EnumeratedTypeWebService`.



```

<%@ WebService Language="C#" Class="EnumeratedTypeWebService" %>

using System;
using System.Web.Services;

public enum GoalType{
    Career = 1,
    Education = 2,
    Travel = 3
}

```

**3** Create a public enumeration for the `GoalType`.

**4** Create three enumerated types.



## Apply It

Instead of returning the enumeration by name, you can return the constant value for the enumeration.

### TYPE THIS:

```
<%@ WebService Language="C#" Class="EnumeratedTypeWebService_ai" %>
using System;
using System.Web.Services;

public enum GoalType{
    Career = 1,
    Educational = 2,
    Travel = 3
}

public class EnumeratedTypeWebService_ai : WebService {

    [WebMethod]
    public int GetGoalTypeConstantValue(GoalType goaltypeMember) {
        return (int) goaltypeMember;
    }
}
```

### RESULT:

This Web Service takes in an enumeration member and returns its constant value. For example, when you test this with "Travel" for the goaltypeMember, the response is 3.



```
Untitled - Notepad
File Edit Format Help
<%@ WebService Language="C#" Class="EnumeratedTypeWebService" %>
using System;
using System.Web.Services;

public enum GoalType{
    Career = 1,
    Education = 2,
    Travel = 3
}

public class EnumeratedTypeWebService : WebService {

    [WebMethod]
    public GoalType GetTravelGoalType() {
        return GoalType.Travel;
    }
}
```

**5** Add a WebMethod that returns the GoalType enumeration.

**6** Return the value for a specific enumeration.

```
http://localhost/EnumeratedTypeWebService.asmx/GetTravelGoalType? - Microsoft Internet Exp...
File Edit View Favorites Tools Help
Back Forward Stop Home Personal Bar Search Favorites
Address http://localhost/EnumeratedTypeWebService.asmx/GetTravelGoalType? Go Links
<?xml version="1.0" ?>
<GoalType xmlns="http://tempuri.org/">Travel</GoalType>
```

**7** Save the file and test the Web Service.

The GoalType element and value appear.

# RETURN AN OBJECT FROM A WEB SERVICE

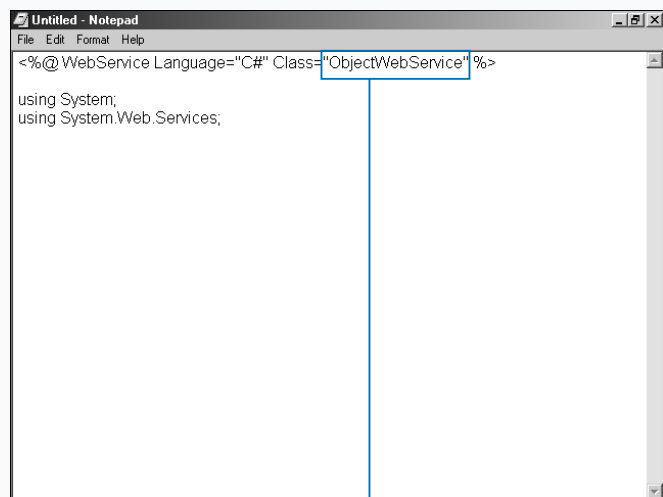
Returning objects from a Web Service enables you to pass very complex return information to the Web Service Client. You can define your own objects and return objects from a Web Service. For example, you can define a `Goal` class and return a `Goal` object from your Web Service.

You can return an object from a Web Service if the Web Service has access to a class declaration. The class declaration defines which members are parts of the class. For example, adding a `GoalId` property to the class and specifying the data type for the property allows for storage of a Goal Identifier. This would be

repeated for other required members of the class. With this class, you can use the name of the class as the return type for the Web Method.

In a Web Method that returns a class, you create a new object as the object type defined in the Web Service file and then populate the properties of the object. This stateful object can then be passed back to the client. The client uses the returned object to obtain values it needs by accessing the object's properties. For more information on working with objects and object properties, see page 44.

## RETURN AN OBJECT FROM A WEB SERVICE



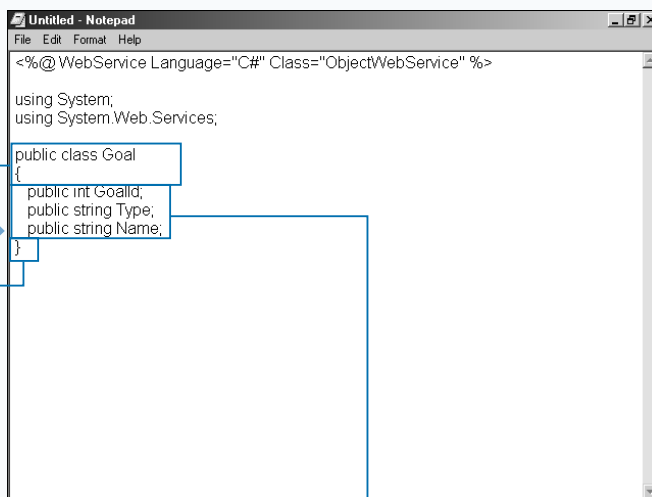
```

Untitled - Notepad
File Edit Format Help
<%@ WebService Language="C#" Class="ObjectWebService" %>

using System;
using System.Web.Services;
  
```

**1** Open the `WebServicesTemplate.aspx` template from the Code Templates directory.

**2** Rename the class to **ObjectWebService**.



```

Untitled - Notepad
File Edit Format Help
<%@ WebService Language="C#" Class="ObjectWebService" %>

using System;
using System.Web.Services;

public class Goal
{
    public int GoalId;
    public string Type;
    public string Name;
}
  
```

**3** Create a public class for goals.

**4** Create `GoalId`, `Type`, and `Name` properties for a goal.

## Apply It

You can determine which instance of an object to return based on an input parameter of the Web Services Method.

### TYPE THIS:

```
<%@ WebService Language="C#" Class="ObjectWebService_ai" %>
using System; using System.Web.Services;

public class Goal{
    public int GoalId; public string Type; public string Name;
}
public class ObjectWebService_ai : WebService {
    [WebMethod] public Goal GetTravelGoal(int intGoalId) {
        Goal goalUser = new Goal();
        switch(intGoalId){
            case 1:
                goalUser.GoalId = 1; goalUser.Type = "Travel";
                goalUser.Name = "Travel to all seven continents";
                break;
            case 2:
                goalUser.GoalId = 2; goalUser.Type = "Travel";
                goalUser.Name = "Travel to Asia";
                break;
            default:
                goto case 2;
        }
        return goalUser;
    }
}
```

### RESULT:

A Web Service that will return an object that has a state that depends on what is passed into the Web Service Method. The return is a stateful Goal object.



```
Untitled - Notepad
File Edit Format Help
<%@ WebService Language="C#" Class="ObjectWebService" %>
using System;
using System.Web.Services;

public class Goal
{
    public int GoalId;
    public string Type;
    public string Name;
}

public class ObjectWebService : WebService {
    [WebMethod]
    public Goal GetTravelGoal() {
        Goal goalUser = new Goal();
        goalUser.GoalId = 1;
        goalUser.Type = "Travel";
        goalUser.Name = "Travel to all seven continents";
        return goalUser;
    }
}
```

**5** Add a WebMethod that returns the Goal class.

**6** Create a new Goal variable.

**7** Set the GoalId, Type, and Name properties.

**8** Return the object.

```
http://localhost/ObjectWebService.asmx/GetTravelGoal? - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/ObjectWebService.asmx/GetTravelGoal?
Go Links
<?xml version="1.0" ?>
- <Goal xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
  xmlns="http://tempuri.org/">
  <GoalId>1</GoalId>
  <Type>Travel</Type>
  <Name>Travel to all seven continents</Name>
</Goal>
```

**9** Save the file and test the Web Service.

The Goal object returns from the Web Service.

# RETURN XML FROM A WEB SERVICE

Returning XML from a Web Service is useful when passing hierarchical data back to a Web Service Client. When passing back XML, you can construct your own XML strings or use the XML framework classes to simplify the construction of the XML. For example, you can build a list of goals along with any important attributes of those goals in an XML document and return this XML from your Web Service.

To return XML from a Web Service, you need to build XML in the Web Service Method. The XML that is built in the Web Service Method can originate from a

variety of sources. You can build XML: a) from scratch by concatenating strings that represent your XML; b) from scratch using the `System.Xml` namespace; c) by loading an existing XML document; d) by transforming an XML document with an XSLT document into a new XML document; or e) by using Microsoft's SQL Server's XML query engine.

This is not an exhaustive list of XML sources, but it gives you a good idea of the possibilities. See pages 146 to 149 for more.

## RETURN XML FROM A WEB SERVICE

```

Untitled - Notepad
File Edit Format Help
<%@WebService Language="C#" Class="XMLWebService" %>

using System;
using System.Web.Services;
using System.Xml;
  
```

**1** Open the `WebServicesTemplate.aspx` template from the CD-ROM.

**2** Rename the class to `XMLWebService`.

**3** Add an alias for the `System.Xml` namespace.

```

Untitled - Notepad
File Edit Format Help
<%@WebService Language="C#" Class="XMLWebService" %>

using System;
using System.Web.Services;
using System.Xml;

public class XMLWebService : WebService {

    [WebMethod]
    public XmlDocument GetTravelGoalType() {

    }

}
  
```

**4** Add a `WebMethod` that returns the `XmlDocument` class.

**Apply It**

You can load an external XML document using the `System.Xml` namespace. For example, the Web Service below returns all Goals in the `goals.xml` document. Notice the use of the `Server.MapPath` function that returns the location of the file on the server.

**TYPE THIS:**

```
<%@ WebService Language="C#" Class="XMLWebService_ai" %>

using System;
using System.Web.Services;
using System.Xml;

public class XMLWebService_ai : WebService {
    [WebMethod]
    public XmlDocument GetAllGoals(){
        XmlDocument xmlDocumentGoals = new XmlDocument();
        xmlDocumentGoals.Load(Server.MapPath("goals.xml"));
        return xmlDocumentGoals;
    }
}
```

**RESULT:**

A Web Service that returns an XML document from the Web server that contains a list of goals.



```
Untitled - Notepad
File Edit Format Help
<%@ WebService Language="C#" Class="XMLWebService" %>

using System;
using System.Web.Services;
using System.Xml;

public class XMLWebService : WebService {

    [WebMethod]
    public XmlDocument GetTravelGoalType() {
        XmlDocument xmlDocumentGoalType = new XmlDocument();
        xmlDocumentGoalType.LoadXml("<GoalType>Travel</GoalType>");
        return xmlDocumentGoalType;
    }
}
```

```
http://localhost/XMLWebService.asmx/GetTravelGoalType? - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/XMLWebService.asmx/GetTravelGoalType?
Go Links

<?xml version="1.0" ?>
- <AnyNode xmlns="http://tempuri.org/">
  <GoalType xmlns="">Travel</GoalType>
</AnyNode>
```

**5** Create a new `XmlDocument` variable.

**6** Load an XML string into the `XmlDocument`.

**7** Return the `XmlDocument` variable.

**8** Save the file and test the Web Service.

The `XmlDocument` returns from the Web Service.

# RETURN SQL DATA FROM A WEB SERVICE

Having the ability to access databases from a Web Service enables you to build applications that require data from various sources. Originating data from a SQL data store is one way to provide interoperability between applications. The SQL data can be central data storage for any application that can connect and issue commands against the SQL data store. This shared data is one way to enable applications to interact with each other.

There are a number of steps that you have to take to provide SQL Data from your Web Service. The first is to add the `SQL.Data` and the `SQL.Data.SqlClient` aliases so you can work with the objects in those

namespaces. The `SQL.Data.SqlClient` is optimized to Microsoft SQL Server databases; if you want access to other SQL data stores, you can reference the `SQL.Data.OleDb` namespace. You can then create a Web Method that will return a `DataSet` object. In the Web Method, you create a connection to the database, retrieve the data using a SQL statement, and return the data that was retrieved.

When creating Web Services that access data, you need to also consider what parameters you need to return the data (for example, primary keys). To learn more about working with data access, see page 126.

## RETURN SQL DATA FROM A WEB SERVICE

```

Untitled - Notepad
File Edit Format Help
<%@ WebService Language="C#" Class="SQLDataWebService" %>

using System;
using System.Data;
using System.Data.SqlClient;
using System.Web.Services;
  
```

**1** Open the `WebServicesTemplate.asmx` template from the Code Templates directory.

**2** Rename the class to `SQLDataWebService`.

**3** Add an alias for the `System.Data` and `System.Data.SqlClient` namespaces.

```

Untitled - Notepad
File Edit Format Help
<%@ WebService Language="C#" Class="SQLDataWebService" %>

using System;
using System.Data;
using System.Data.SqlClient;
using System.Web.Services;

public class SQLDataWebService {

    [WebMethod]
    public DataSet GetBusinessTitles() {
        SqlConnection sqlconnectionPubs = new
        SqlConnection("server=(local)\NetSDK;uid=QUser;pwd=QSPassword;database=pubs");
    }
}
  
```

**4** Add a `WebMethod` that returns the `DataSet` class.

**5** Create a new `SqlConnection` object to connect to the database and initialize the connection with the connection string to connect to the Pubs database.

## Apply It

The task returned the business titles as a Web Service. You can also provide a Web Service that takes in the type of book and returns the SQL data for that type of book.

## TYPE THIS:

```
<%@ WebService Language="C#" Class="SQLDataWebService_ai" %>
using System;
using System.Data;
using System.Data.SqlClient;
using System.Web.Services;

public class SQLDataWebService_ai {
    [WebMethod] public DataSet GetTitles(string stringTitleType) {
        stringTitleType = stringTitleType.ToLower();
        if (stringTitleType != "trad_cook" &
            stringTitleType != "mod_cook" & stringTitleType != "business")
            stringTitleType = "business";
        string stringSQLStatement = "select title, notes, price " +
            "from titles where type='" + stringTitleType + "'";
        SqlConnection sqlconnectionPubs = new SqlConnection
            ("server=(local)\NetSDK;uid=QUser;pwd=QSPassword;" +
            "database=pubs");
        SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter
            (stringSQLStatement, sqlconnectionPubs);
        DataSet datasetTitles = new DataSet();
        sqldataadapterTitles.Fill(datasetTitles, "Titles");
        return datasetTitles;
    }
}
```

## RESULT:

This produces a Web Service that returns a DataSet for the type of title requested.



**6** Create a new `SqlConnection` variable that uses the `SqlConnection` object and a SQL string for retrieving business titles.

**7** Create a `DataSet` object.

**8** Fill the `DataSet` object using the `SqlDataAdapter` object.

**9** Return the `DataSet` object.

**10** Save the file and test the Web Service.

The SQL data returns from the Web Service.

Note the SQL data in the response.

# WORK WITH THE SESSION OBJECT IN A WEB SERVICE

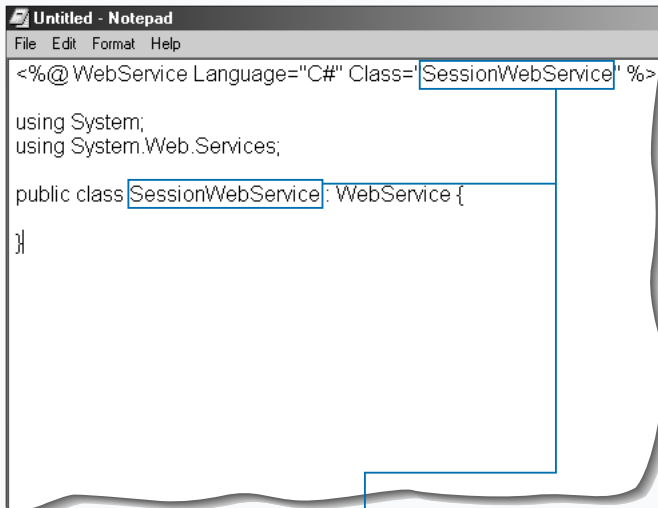
Working with the `Session` object in a Web Service gives your Web Services the capability to have variables that can be used across different requests to Web Services during the same user session.

The process for using the `Session` object is simple. When you declare the Web Method that uses `Session`, you need to specify that `Session` is enabled. You do this by adding `(EnableSession = true)` just after `WebMethod`. By default, `WebMethods` do not have `Session` enabled. After enabling the session, you can access the `Session` object.

You can use `Session` to track state from page to page requests for a particular user. You can work with the `Session` object in different states such as `New Session`, `Existing Session`, and `Abandoned Session`.

You can use `New session` when the user does not have an existing valid session. `New session` enables you to initialize the `Session` object by setting any `Session` variables to any initial value. You can use `Existing Session` on subsequent requests to update or access `Session` variables. You can use `Abandoned Session` when the `Session` times out or you abandon the session. You will need to perform any cleanup necessary that is associated with the `Session` object. For more information on working with the `Session` object, see page 226.

## WORK WITH THE SESSION OBJECT IN A WEB SERVICE



```

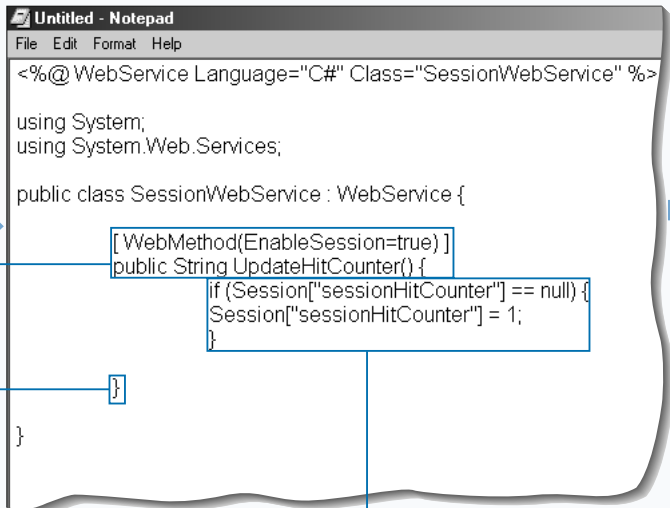
Untitled - Notepad
File Edit Format Help
<%@ WebService Language="C#" Class="SessionWebService" %>

using System;
using System.Web.Services;

public class SessionWebService : WebService {
}
  
```

**1** Open the `WebServicesTemplate.asmx` template from the Code Templates directory.

**2** Rename the class to `SessionWebService`.



```

Untitled - Notepad
File Edit Format Help
<%@ WebService Language="C#" Class="SessionWebService" %>

using System;
using System.Web.Services;

public class SessionWebService : WebService {
    [WebMethod(EnableSession=true)]
    public String UpdateHitCounter() {
        if (Session["sessionHitCounter"] == null) {
            Session["sessionHitCounter"] = 1;
        }
    }
}
  
```

**3** Add a `WebMethod` that returns a string variable with the `Session` state enabled.

**4** Add an `if` statement to initialize the `Session` variable.



## Apply It

With the session Web Service, you can retain state across requests to a Web Service.

### TYPE THIS:

```
<%@ WebService Language="C#" Class="SessionWebService_ai" %>
using System;
using System.Web.Services;

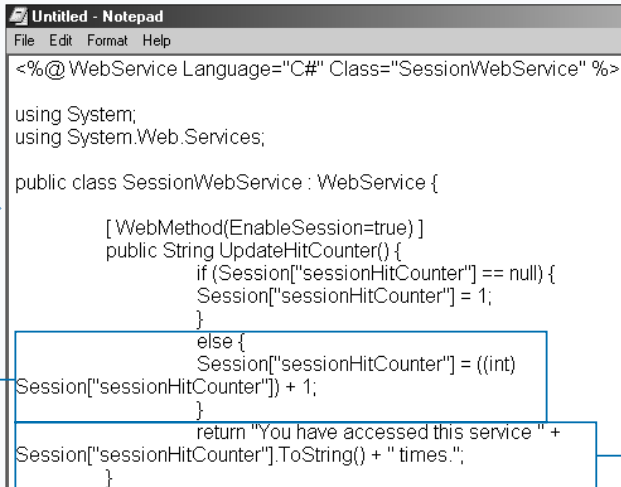
public class SessionWebService_ai : WebService {

    [ WebMethod(EnableSession=true) ]
    public String RememberName(string stringName) {
        string stringPreviousName = "";
        string stringCurrentName = "";

        if (Session["sessionName"] == null) {
            stringPreviousName = "Null";
        }
        else {
            stringPreviousName = Session["sessionName"].ToString();
        }
        Session["sessionName"] = stringName;
        stringCurrentName = stringName;
        return "The previous value for the session variable was " +
            stringPreviousName + ". The new value for the session variable is "
            + stringCurrentName + ".";
    }
}
```

### RESULT:

This produces a Web Service that returns information on the current request and preceding request.



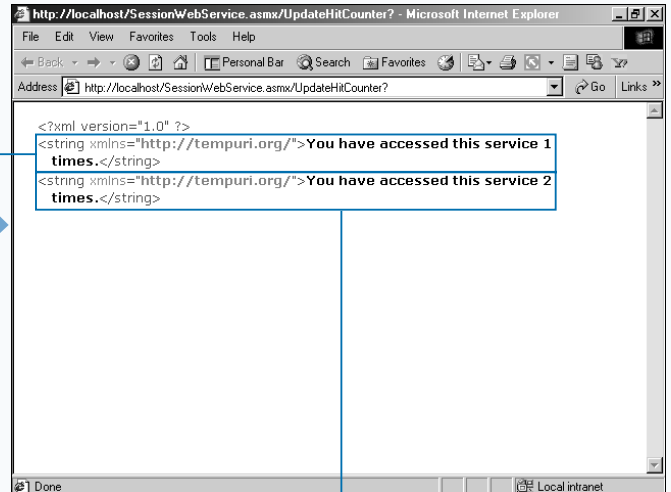
```
using System;
using System.Web.Services;

public class SessionWebService : WebService {

    [ WebMethod(EnableSession=true) ]
    public String UpdateHitCounter() {
        if (Session["sessionHitCounter"] == null) {
            Session["sessionHitCounter"] = 1;
        }
        else {
            Session["sessionHitCounter"] = ((int)
Session["sessionHitCounter"]) + 1;
        }
        return "You have accessed this service " +
Session["sessionHitCounter"].ToString() + " times.";
    }
}
```

**5** Add an **else** statement to increment the **Session** variable by one.

**6** Return a formatted message to the user using the **Session** variable.



**7** Save the file and test the Web Service.

The Web Service returns a message about the number of times the service has been accessed.

**8** Refresh the Web page.

The number of times the service has been accessed is incremented.

# WORK WITH THE APPLICATION OBJECT IN A WEB SERVICE

**Y**ou can work with the `Application` object in a Web Service to enable your Web Services to use variables across all requests to Web Services.

The `Application` object does not require `Session` to be enabled, so you can either leave the Web Method definition as the default or add the `EnableSession=false` statement to explicitly disable `Session`. `Application` data is available in your ASP.NET applications.

In many ways, working with the `Application` object is similar to working with the `Session` object. You have the ability to initialize `Application` variables when the `Application` object is accessed and there

is not an existing `Application` object. You can access or update `Application` variables each time a page is requested. You also have the ability to clean up objects or other memory when the `Application` shuts down.

You want to be aware that all issues associated with the `Application` object are applicable when using the `Application` object in Web Services. For example, you need to lock `Application` variables to ensure the serial access to `Application` variables. To learn more about working with the `Application` object, see page 222.

## WORK WITH THE APPLICATION OBJECT IN A WEB SERVICE

```

Untitled - Notepad
File Edit Format Help
<%@WebService Language="C#" Class="ApplicationWebService" %>
using System;
using System.Web.Services;

public class ApplicationWebService : WebService {
}
  
```

**1** Open the `WebServicesTemplate.aspx` template from the Code Templates directory.

**2** Rename the class to `ApplicationWebService`.

```

Untitled - Notepad
File Edit Format Help
<%@WebService Language="C#" Class="ApplicationWebService" %>
using System;
using System.Web.Services;

public class ApplicationWebService : WebService {
    [WebMethod(EnableSession=false)]
    public String UpdateAppCounter() {
        if (Application["applicationHitCounter"] == null) {
            Application["applicationHitCounter"] = 1;
        }
    }
}
  
```

**3** Add a `WebMethod` that returns a string variable with the `Session` state disabled.

**4** Add an `if` statement to initialize the `Application` variable.

## Apply It

To prevent corruption of your `Application` variables, it is a good idea to lock the `Application` variable before you update it.

### TYPE THIS:

```
<%@ WebService Language="C#" Class="ApplicationWebService_ai" %>
using System;
using System.Web.Services;

public class ApplicationWebService_ai : WebService {
    [ WebMethod(EnableSession=false)]
    public String UpdateAppCounter() {
        if (Application["applicationHitCounter"] == null) {
            Application.Lock();
            Application["applicationHitCounter"] = 1;
            Application.Unlock();
        }
        else {
            Application.Lock();
            Application["applicationHitCounter"] =
                ((int) Application["applicationHitCounter"]) + 1;
            Application.Unlock();
        }
        return "This service has been accessed " +
            Application["applicationHitCounter"].ToString() +
            " times.";
    }
}
```

### RESULT:

This produces a Web Service that returns the current hit count of a Web Service. Note that the updating of the `Application` variable, which keeps track of the count, is locked during the update.

```
Untitled - Notepad
File Edit Format Help
<%@ WebService Language="C#" Class="ApplicationWebService" %>
using System;
using System.Web.Services;

public class ApplicationWebService : WebService {
    [ WebMethod(EnableSession=false)]
    public String UpdateAppCounter() {
        if (Application["applicationHitCounter"] == null) {
            Application["applicationHitCounter"] = 1;
        }
        else {
            Application["applicationHitCounter"] = ((int)
Application["applicationHitCounter"]) + 1;
        }
        return "This service has been accessed " +
Application["applicationHitCounter"].ToString() + " times.";
    }
}
```

**5** Add an `else` statement to increment the `Application` variable by one.

**6** Return a formatted message to the user using the `Application` variable.

```
http://localhost/ApplicationWebService.asmx/UpdateApplicationCounter? - Microsoft Internet Ex...
File Edit View Favorites Tools Help
Back Forward Stop Home Personal Bar Search Favorites
Address http://localhost/ApplicationWebService.asmx/UpdateApplicationCounter? Go Links
<?xml version="1.0" ?>
<string xmlns="http://tempuri.org/">This service has been accessed 1
times.</string>
<string xmlns="http://tempuri.org/">This service has been accessed 2
times.</string>
Done Local intranet
```

**7** Save the file and test the Web Service.

The Web Service returns a message about the number of times the service has been accessed.

**8** Refresh the Web page.

The number of times the service has been accessed is incremented.

# CREATE A CLIENT WEB PAGE FOR A WEB SERVICE

To consume a Web Service, you can create Web Service Clients. ASP.NET framework creates clients for you automatically if you access the Web Service file (\*.asmx) directly. For your custom applications, you need to create your own client to call Web Services.

You need to walk through a couple of steps to enable a Client Web Page. The first is to create a service definition file. This can be created using the `disco` command and passing the URL to the Web Service. This will create a service definition file. You then need to use this service definition file to create a proxy

class for the Web Class. You use the `wsdl` command to pass the service definition file name to it. The result of running this command is a proxy class that you now need to compile to the bin directory as a library. You can use the `csc` command to compile the class.

After you compile the class, you can then import the namespace of the Web Service into the Web Page that will consume the Web Service. On the client Web page, you create an instance of the Web Service just like any other .NET object. After you create an instance, you can call methods from the Web Service.

## CREATE A CLIENT WEB PAGE FOR A WEB SERVICE

```

C:\WINNT\System32\cmd.exe
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\inetpub\wwwroot>
C:\inetpub\wwwroot>wsdl http://localhost/SimpleWebService.asmx?WSDL
Microsoft (C) Web Services Description Language Utility
[Microsoft .NET Framework Version 1.0.2615.1]
Copyright (C) Microsoft Corp 2000. All rights reserved.

C:\inetpub\wwwroot>SimpleWebService.cs
Writing file 'C:\inetpub\wwwroot\SimpleWebService.cs'.

C:\inetpub\wwwroot>csc /out:bin\SimpleWebService.dll /t:library /r:System.
Web.Services.dll /r:System.Web.Services.dll SimpleWebService.cs
Microsoft (R) Visual C# Compiler Version 7.00.2148 [CLR version v1.0.2615]
Copyright (C) Microsoft Corp 2000. All rights reserved.

C:\inetpub\wwwroot>_
  
```

```

<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE="Verdana">
<FORM RUNAT="Server">
<H3><ASP.LABEL ID="labelMessage" RUNAT="Server" /></H3>
</FORM>
</FONT>
</BODY>
</HTML>
  
```

- 1 Open the command prompt.
- 2 Change directories to where the Web Service is located.

- 3 Run the `wsdl` command to create a proxy class for the Web Service.
- 4 Compile the proxy class to the `/bin` directory.

■ The proxy class is created and compiled.

- 5 Open `GenericTemplate.aspx` from the Code Templates directory.

- 6 Add a form to the page.
- 7 Add a `Label` control to the form.

## Apply It

You can also pass data to a Web Service from a control on a form on a Web page. To run this code, you need to run the `ClientWebService_ai.bat` that creates and compiles the proxy classes for this code.

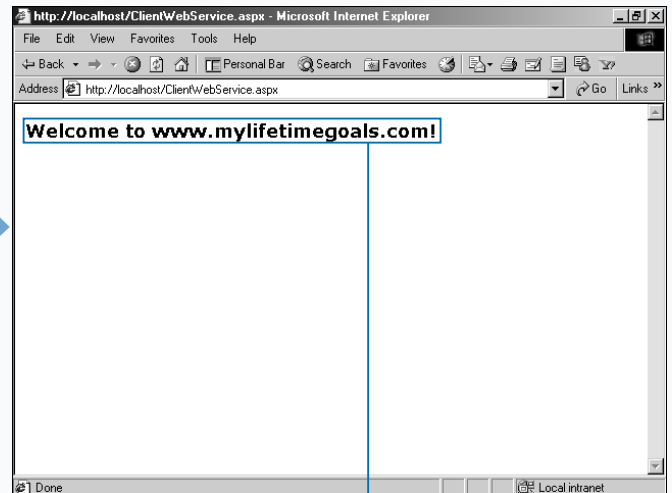
### TYPE THIS (IN THE SERVER SCRIPT BLOCK):

```
protected void Page_Load(Object Src, EventArgs E){
    if (!Page.IsPostBack){
        SimpleWebService simplewebserviceMessage = new SimpleWebService();
        string stringMessage = simplewebserviceMessage.SayWelcomeStatement();
        labelMessage.Text = stringMessage;}
    else{
        ParameterWebService parameterwebserviceMessage = new ParameterWebService();
        string stringMessage =
            parameterwebserviceMessage.SayWelcomeStatement (textboxName.Text);
        labelMessage.Text = stringMessage;    }}}
```

### RESULT:

A Web page displays the greeting. If you submit the page with your name, it will call a different Web Service that customizes greeting with your name in it.

```
Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
protected void Page_Load(Object Src, EventArgs E){
    SimpleWebService simplewebserviceMessage = new SimpleWebService();
    string stringMessage = simplewebserviceMessage.SayWelcomeStatement();
    labelMessage.Text = stringMessage;
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<FORM RUNAT="Server">
<H3><ASP:LABEL ID="labelMessage" RUNAT="Server" /></H3>
</FORM>
</FONT>
</BODY>
</HTML>
```



**8** Add the `Page_Load` function to the page.

**9** Create a new instance of `SimpleWebService`.

**10** Set the result of calling the `SayWelcomeMessage` from the `SimpleWebService` into a string variable.

**11** Update the `Label` control with the result.

**12** Save the file and request it from the Web server.

A welcome message from the Web Service appears.

# CREATE A CLIENT CONSOLE APPLICATION FOR A WEB SERVICE

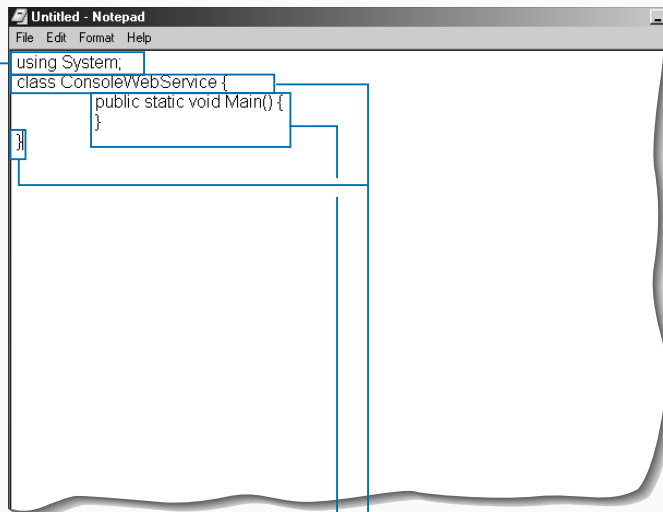
**Y**ou can use a Console Application as a test harness for a Web Service. ASP.NET Web Services can be easily consumed by WinForm Applications or Console Applications.

Accessing a Web service from a console application requires many of the same steps as accessing it from a Web page. The first step is to create a service definition file by using the `disco` command and passing the URL to the Web Service. This will create a service definition file. You then need to use this service definition file to create a proxy class for the Web Class. You can use the `wsdl` command to pass

the service definition file name to it. The result of running this command is a proxy class that you now need to compile to the bin directory as a library. You can use the `csc` command to compile the class.

After you have compiled the class, you can then import the namespace of the Web Service into the Console Application that will consume the Web Service. Within the Console Application, you can create an instance of the Web Service. After you have created an instance, you can call methods from the Web Service and display results to the console.

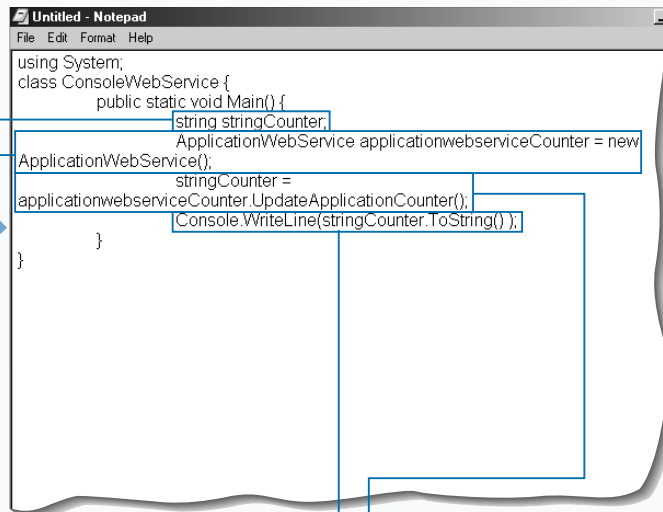
## CREATE A CLIENT CONSOLE APPLICATION FOR A WEB SERVICE



```

Untitled - Notepad
File Edit Format Help
using System;
class ConsoleWebService {
    public static void Main() {
    }
}
  
```

- 1 Open a new document in your text editor.
- 2 Import the `System` namespace.
- 3 Create a new class.
- 4 Create the `Main` function.



```

Untitled - Notepad
File Edit Format Help
using System;
class ConsoleWebService {
    public static void Main() {
        string stringCounter;
        ApplicationWebService applicationwebserviceCounter = new
        ApplicationWebService();
        stringCounter =
        applicationwebserviceCounter.UpdateApplicationCounter();
        Console.WriteLine(stringCounter.ToString());
    }
}
  
```

- 5 Create an integer variable in the `Main` function.
- 6 Create a new instance of `ApplicationWebService`.
- 7 Set the result of calling the `UpdateApplicationCounter` from the `ApplicationWebService` into an integer variable.
- 8 Format and write the result to the command line.

## Apply It

You can also pass data to the Web Service from the Console Application via a command line parameter. To run this code, you need to run the `ConsoleWebService_ai.bat` that creates and compiles a proxy class and Web Service Client for this code.

### TYPE THIS:

```
using System;

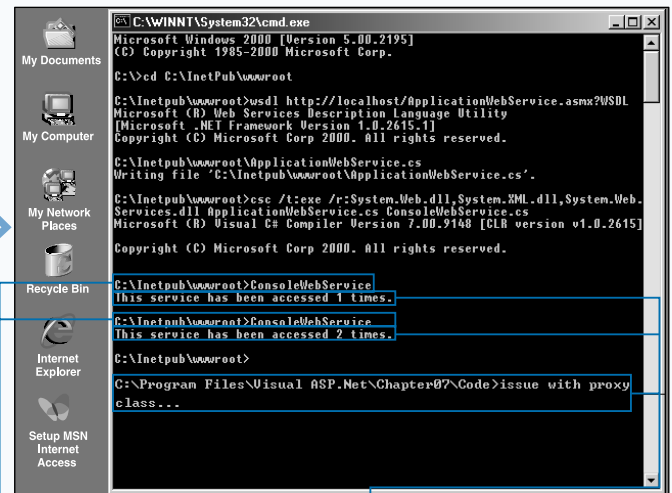
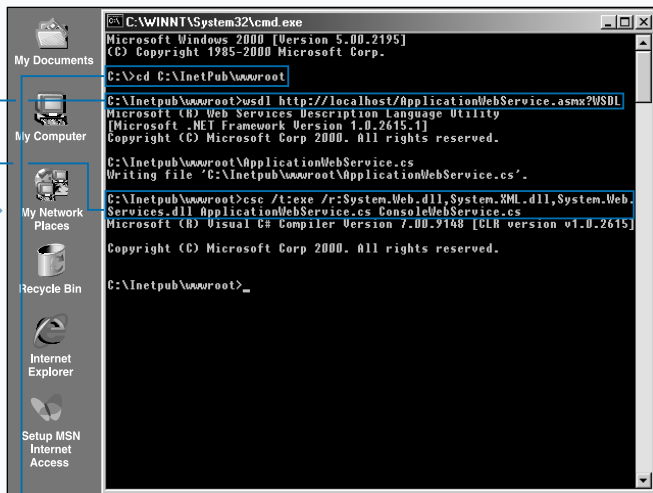
class ConsoleWebService{
    public static void Main(string[] args) {
        int intInitialize = Convert.ToInt32(args[0]);
        if (intInitialize >= 0) {
            ApplicationWebService_ai
            applicationwebservicecounter = new
            ApplicationWebService_ai();

            string strHitCountMessage =
            applicationwebservicecounter.
            UpdateApplicationCounter(intInitialize);
            Console.WriteLine(strHitCountMessage);
        }
    }
}
```

### RESULT:

```
C:\>ConsoleWebService_ai.bat
C:\>ApplicationWebService_ai.exe 10

This service has been accessed 10 time(s).
```



- 9 Open the command prompt.
- 10 Change directories to where the Web Service is located.
- 11 Run the `wsdl` command to create a proxy class for the Web Service.

- 12 Compile the console class and the Web Service.
  - The proxy class is created and compiled and the console application is compiled.

- 13 Run the created executable a couple of times.

- A message from the Web Service appears about the number of times the Web Service has been accessed.

# CREATE A SIMPLE COMPONENT

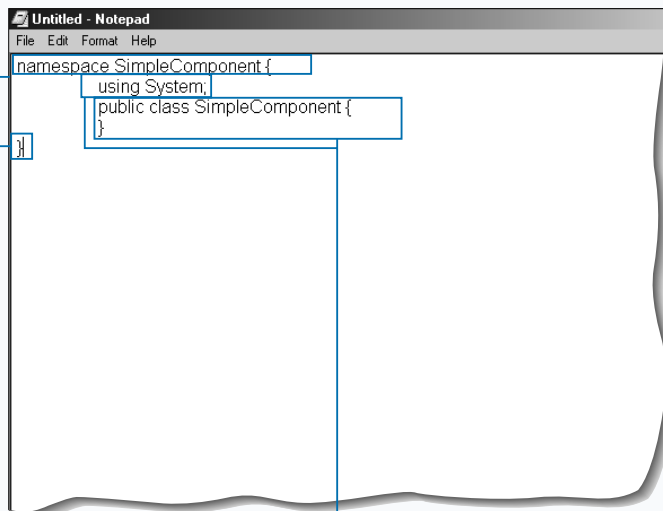
Components enable you to encapsulate business logic that can be reused across several applications. You can create managed classes in C# or any other .NET compliant language to create your reusable components.

The process for creating a simple component starts with creating a C# source file. In this source file you first declare the namespace for the classes contained in the source file. In your C# source file, you need to add functions to hold your business logic. For instance, in the simple component example you have a function, called `SayWelcomeStatement`, that returns the same string message to any caller. Note as

well that you create an alias to the `System` namespace in the C# source file.

When you finish creating the code, you need to compile the program using the `csc` command at the command prompt. You use this compiler to create a DLL with your source code. To use the component on Web pages in your site, you need to place the compiled DLL in the `/bin` directory of the Web site. To reference the component, you need to import the namespace that you specified in the source file. After you have done this, you can create an instance of the class created and call functions in that class.

## CREATE A SIMPLE COMPONENT



```

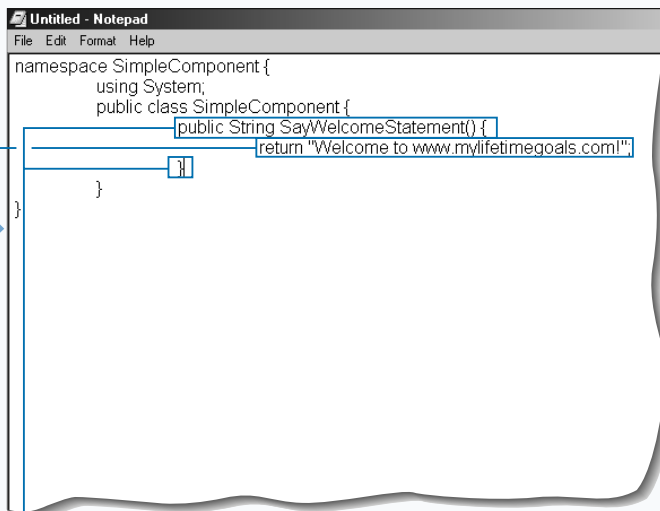
Untitled - Notepad
File Edit Format Help
namespace SimpleComponent {
    using System;
    public class SimpleComponent {
    }
}
  
```

**1** Open a new document in your text editor.

**2** Create a new namespace.

**3** Add an alias to the `System` namespace.

**4** Create a `public` class.



```

Untitled - Notepad
File Edit Format Help
namespace SimpleComponent {
    using System;
    public class SimpleComponent {
        public String SayWelcomeStatement() {
            return "Welcome to www.mylifetimegoals.com!";
        }
    }
}
  
```

**5** Create a `public` function that returns a string variable.

**6** Return a message to the caller.

**7** Save the file.

*Note: In this example, the file is being saved to the default Web site location at `C:\inetpub\wwwroot`.*



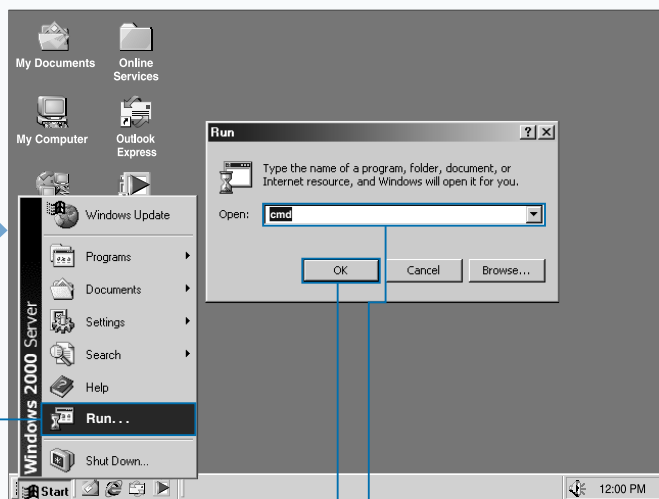
**Extra**

Aliases enable you to reference classes without full qualification of the class. Aliases are set by placing the `using` keyword before a namespace. Using aliases can help reduce the length of your code, making it easier to read.

**Example:**

```
Using Transformer = system.xml. xsl.xsltransform;
```

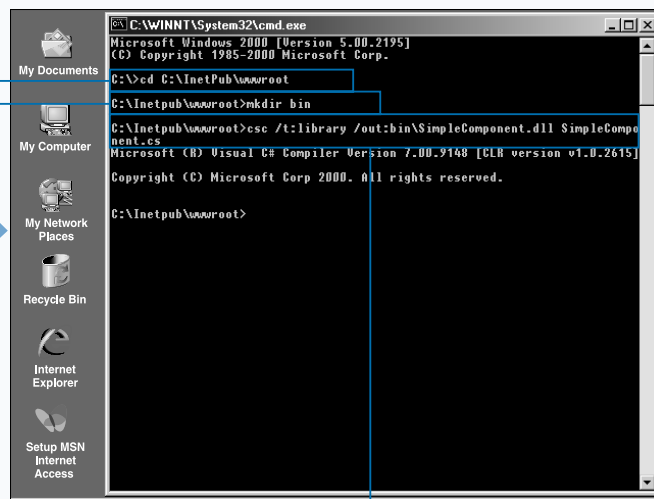
Batch files are very useful to take care of repetitive tasks like compiling a component. For example, the C# components for this chapter all come with batch files on the CD-ROM that have the `csc` commands for compiling the component. Look in the `Code` directory for this chapter and find a batch file with the same name as the component you create in the task (for example, `SimpleComponent.bat` is used to create `SimpleComponent`). Open this file with a text editor and find the following source: `csc /t:library /out:bin\SimpleComponent.dll SimpleComponent.cs`. You can use these batch files by simply typing in the name of the component (for example, `SimpleComponent`) while at the command prompt. Note: You must navigate to the directory where the batch file and the component source code are located.



**8** Click Start ⇨ Run to open the dialog box.

**9** Type **cmd** in the Open field.

**10** Click OK to open the command prompt.



**11** Change directories to where you saved the source file by using the `cd` command.

**12** Create a `/bin` directory for your compiled libraries.

**13** Use the `csc` command to compile the class at the command prompt.

*Note: See page 34 for instructions on compiling.*

CONTINUED

# CREATE A SIMPLE COMPONENT

Components enable you to create distributed, reusable architectures. If you put your business logic and data access into components, you put yourself in a better situation for addressing application development challenges. The challenges can be issues with security, scalability, performance, stability, or reusability.

In terms of reusability, when you create components in .NET with managed code, you need to decide if the component is part of a private assembly or a global assembly. In many cases, you put components into private assemblies. This is the simplest way to create, manage, and use components. No special registration process is needed for a private assembly, except for

putting the compiled DLL in the `/bin` directory of your Web site. Using private assemblies for your components makes it very simple to deploy ASP.NET applications. All you need to do is `xcopy` the files to a Web server.

Global assemblies entail more detail. A global assembly needs to be put in the Global Assembly Cache (GAC). This is required to register a global assembly. If the component is registered in the global assembly, then you do not have the ability to just `xcopy` the files for moving a Web site. You will also need to incorporate a registration process for your global assemblies.

## CREATE A SIMPLE COMPONENT (CONTINUED)

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="SimpleComponent" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
public void Page_Load(Object sender, EventArgs E){
SimpleComponent simplecomponentMessage = new SimpleComponent();
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
</FONT>
</BODY>
</HTML>

```

**14** Open the `GenericTemplate.aspx` template from the Code Templates directory.

**15** Import the `SimpleComponent` namespace.

**16** Add the `Page_Load` function.

**17** Create a new variable of type `SimpleComponent`.

```

Untitled - Notepad
File Edit Format Help
<BODY>
<FONT FACE ="Verdana">
<%@ Import Namespace="SimpleComponent" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
public void Page_Load(Object sender, EventArgs E){
SimpleComponent simplecomponentMessage = new SimpleComponent();
string stringMessage = simplecomponentMessage.SayWelcomeStatement();
labelMessage.Text = stringMessage;
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<FORM RUNAT="Server">
<H3><ASP:LABEL ID="labelMessage" RUNAT="Server" /></H3>
</FORM>
</FONT>
</BODY>
</HTML>

```

**18** Create a new string variable and read the result of `SayWelcomeStatement` into that string.

**19** Set the label on the page equal to what was returned from `SayWelcomeStatement`.

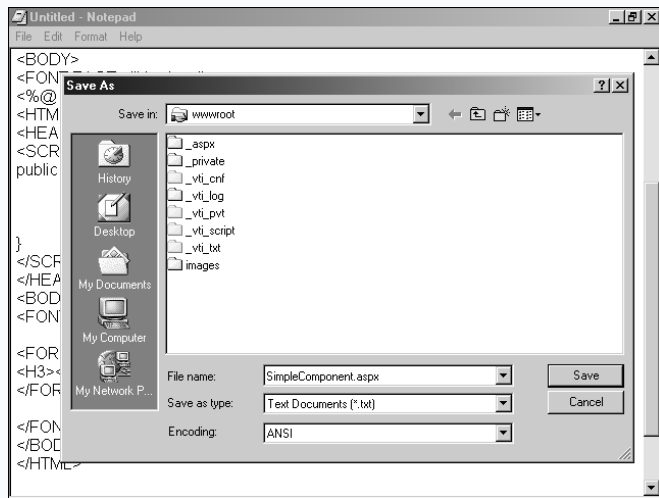
**20** Add a server form.

**21** Add a label to the server form.

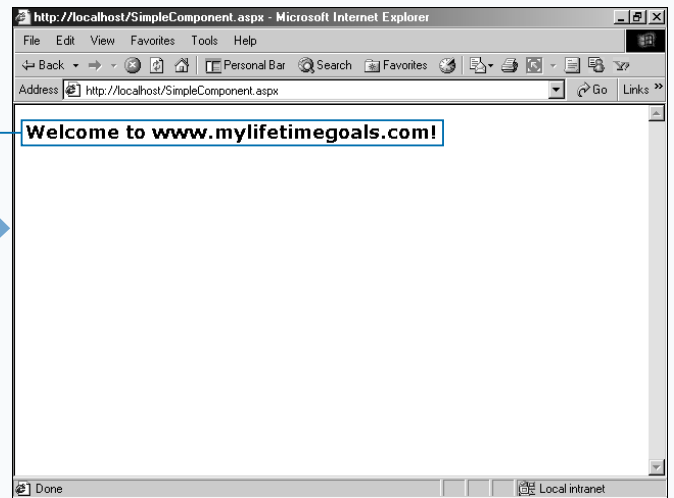
**Extra**

You can package up multiple components into deployable units called assemblies. Assemblies are how the .NET Framework manages components for: deployment, version control, reuse, activation scoping, and security permissions. When creating an assembly, you need to decide on whether you want the assembly to be private or global. There are pros and cons to either choice. The main benefit for choosing private assemblies is the ease of use, especially with deployment. Private assemblies support the `xcopy` deployment, which is not available for COM components.

To support some of the advanced features of .NET components, like sharing a component across multiple applications, you will need to create a global assembly. Each computer where the common language runtime is installed has a machine-wide code cache called the global assembly cache. You should share assemblies by installing them into the global assembly cache only when you need to. Typically, for ASP.NET applications you will create private assemblies and put them into the `/bin` directory of your Web site.



**22** Save the file and request it from the Web server.



A welcome message appears.

# CREATE A STATEFUL COMPONENT

For most cases, your applications only use components that do not hold any state between calls to the component (stateless components). If you have an application that makes multiple calls to a component and these calls rely on common state, then you will want to have attributes on your classes to hold this state. The state can be provided by setting properties on the class programmatically on your Web page or state can be set in the initialization of the class.

Like a simple component, the process for creating a stateful component starts with creating a C# source file. You first declare the namespace and then add any aliases that you need. For example, if you want to

build strings, you can add an alias for the `System.Text` namespace to get access to the `StringBuilder` class.

To create a property for the class, you first create a private variable. Next, you will want to add some code for initializing the variable when the class is created. Then, you will need to create a public variable that will be used to read and write to the private variable. After you finish creating the code to create the property, you need to compile the program using the `csc` command at the command prompt. Finally, you need to create the Web page that uses this component.

## CREATE A STATEFUL COMPONENT

```

Untitled - Notepad
File Edit Format Help
namespace StatefulComponent {
    using System;
    using System.Text;
    public class StatefulComponent {
    }
}
  
```

**1** Open a new document in your text editor.

**2** Create a new namespace.

**3** Add an alias to the `System` and `System.Text` namespace.

**4** Create a public class.

```

Untitled - Notepad
File Edit Format Help
namespace StatefulComponent {
    using System;
    using System.Text;
    public class StatefulComponent {
        private String _name;
        public StatefulComponent() {
            _name = null;
        }
        public String Name {
            get {
                return _name;
            }
            set {
                _name = value;
            }
        }
        public String SayWelcomeStatement() {
        }
    }
}
  
```

**5** Create a private string variable for holding state.

**6** Initialize the private string variable when an instance of the class is created.

**7** Add a public string variable that has a method to read the value of the private string variable and a method to write to the private string variable.

**8** Add a function that returns a string.

## Apply It

You can create a Web page containing a form used for updating the Name property of the stateful component.

### TYPE THIS:

```
<%@ Import Namespace="StatefulComponent" %>
<HTML>
<HEAD><SCRIPT LANGUAGE="C#" RUNAT="Server">
public void Page_Load(Object sender, EventArgs E){
    if (!Page.IsPostBack){
        StatefulComponent statefulcomponentMessage = new StatefulComponent();
        string stringMessage = statefulcomponentMessage.SayWelcomeStatement();
        labelMessage.Text = stringMessage;
    }
    else{
        StatefulComponent statefulcomponentMessage = new StatefulComponent();
        statefulcomponentMessage.Name = textboxName.Text;
        string stringMessage = statefulcomponentMessage.SayWelcomeStatement();
        labelMessage.Text = stringMessage;
    }
}
</SCRIPT></HEAD>
<BODY>
<FONT FACE ="Verdana"><FORM RUNAT="Server">
<H3><ASP:LABEL ID="labelMessage" RUNAT="Server" /></H3>
<P><ASP:TEXTBOX ID="textboxName" TEXT="Type your name here."
RUNAT="Server" WIDTH="300px"/></P>
<ASP:BUTTON ID="buttonPersonalize" RUNAT="Server" TEXT="Personalize"/>
</FORM></FONT></BODY></HTML>
```

### RESULT:

A Web page allows you to put your name in a text box that is used to give you a personal greeting when the form is posted back to the server.

```
public String Name {
    get {
        return _name;
    }
    set {
        _name = value;
    }
}
public String SayWelcomeStatement() {
    StringBuilder stringBuilderMessage = new
Stringbuilder("Welcome to www.mylifetimegoals.com");
    if (_name != null){
        stringBuilderMessage.Append(" ");
        stringBuilderMessage.Append(_name);
        stringBuilderMessage.Append("!");
    }
    else{
        stringBuilderMessage.Append("!");
    }
    return stringBuilderMessage.ToString();
}
```

**9** Create a **StringBuilder** variable and initialize with a message.

**10** Use an **if** statement to customize the message to the user if the **Name** property is set.

**11** Use an **else** statement to handle the case where the **Name** property has not been set.

**12** Return the **StringBuilder** variable.

```
C:\WINNT\System32\cmd.exe
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\>cd C:\inetpub\wwwroot
C:\inetpub\wwwroot>csc /t:library /out:bin\StatefulComponent.dll StatefulComponent.cs
Microsoft (R) Visual C# Compiler Version 7.00.9148 [CLR version v1.0.2615]
Copyright (C) Microsoft Corp 2000. All rights reserved.

C:\inetpub\wwwroot>
```

**13** Go to the command prompt.

**14** Compile the component using the **csc** command.

CONTINUED

# CREATE A STATEFUL COMPONENT

You need to be cautious when using stateful components on a Web page. You need to understand that Web applications are by default a stateless model. This is one of the toughest programming challenges when moving from Win32 applications to Web applications on the Windows platform.

When a component is created for a Web page or Web Service, the state will only be available during the lifetime of the user's request of the resource (Web page or Web Service). When the request is done, the

components used in the Web page or Web Service are released to .NET's garbage collection. The .NET Framework's garbage collector manages the allocation and release of memory for your application. Because this is how the Web server operates, you do not put state that needs to be held across pages in the standard components used by your Web application. There are mechanisms built into the .NET framework to enable you to manage state across pages in a site. The `Session` object is one common mechanism that is available to you. See page 210 for details on state management.

## CREATE A STATEFUL COMPONENT (CONTINUED)

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="StatefulComponent" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
public void Page_Load(Object sender, EventArgs E) {
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

</FONT>
</BODY>
</HTML>
  
```

**15** Open the `GenericTemplate.aspx` template from the Code Templates directory.

**16** Import the `StatefulComponent` namespace.

**17** Add the `Page_Load` function.

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="StatefulComponent" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
public void Page_Load(Object sender, EventArgs E) {
    StatefulComponent statefulcomponentMessage = new
    StatefulComponent();
    statefulcomponentMessage.Name = "Danny";
    string stringMessage =
    statefulcomponentMessage.SayWelcomeStatement();
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

</FONT>
</BODY>
</HTML>
  
```

**18** Create a new variable of type `StatefulComponent`.

**19** Set the `Name` property for the `StatefulComponent`.

**20** Create a new string variable and read the result of `SayWelcomeStatement` into that string.

**Extra**

You can control your stateful components that persist data with fields and properties by initializing them with object constructors. You can also control the assignment of properties with validation code.

**Example:**

```
using System;
public class Goal
{
    private String m_strDescription;
    public Goal()
    {
        m_strDescription = null;
    }
    public String Name
    {
        get
        {
            return m_strDescription;
        }
        set
        {
            if (value.Length < 30)
                m_strDescription = value;
        }
    }
}
```

```
Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="StatefulComponent" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
public void Page_Load(Object sender, EventArgs E) {
    StatefulComponent statefulcomponentMessage = new
    StatefulComponent();
    statefulcomponentMessage Name = "Danny";
    string stringMessage =
    statefulcomponentMessage.SayWelcomeStatement();
    labelMessage.Text = stringMessage;
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<FORM RUNAT="Server">
<H3><ASP.LABEL ID="labelMessage" RUNAT="Server" /></H3>
</FORM>
</FONT>
</BODY>
</HTML>
```

**21** Set the label on the page equal to what was returned from `SayWelcomeStatement`.

**22** Add a server form.

**23** Add a label to the server form.

```
http://localhost/StatefulComponent.aspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Home Personal Bar Search Favorites
Address http://localhost/StatefulComponent.aspx Go Links
Welcome to www.mylifetimegoals.com Danny!
Done Local intranet
```

**24** Save the file and request it from the Web server.

A welcome message appears that displays the value set for the `Name` property.

# CREATE A TWO-TIER WEB FORM

**Y**ou can abstract your data access from your Web page by putting the data access code into a component. This gives you separation between presentation and data. This separation is useful if you want to have the flexibility of changing the database or data access without having to rewrite your presentation code and HTML.

The first step to creating a Web Application that is split into two tiers is to create a Data Access Layer. To create a data access layer, start by creating a C# source file. In this source file, you add code related to connecting to your data source. See page 126 for

detail on how to program with ADO.NET (.NET framework classes that are used for data access). After you have a connection to the database, you need to add functions to retrieve data from the data source. This can be done with SQL Select statements or stored procedures.

When you are done creating the component, you can compile it to the /bin directory as a library and use it on a Web page.

## CREATE A DATA LAYER

```

Untitled - Notepad
File Edit Format Help
namespace DataLayer {
    using System;
    using System.Data;
    using System.Data.SqlClient;
    public class DataObject {
        private String _stringConnection;
    }
}

```

- 1 Open a new document in your text editor.
- 2 Create a new namespace.
- 3 Add an alias to the `System`, `System.Data`, and `System.Data.SqlClient` namespaces.

- 4 Create a `public` class.
- 5 Create a `private string` variable for holding state.

```

Untitled - Notepad
File Edit Format Help
namespace DataLayer {
    using System;
    using System.Data;
    using System.Data.SqlClient;
    public class DataObject {
        private String _stringConnection;
        public DataObject() {
            _stringConnection = null;
        }
        public DataObject(String connStr) {
            _stringConnection = connStr;
        }
        public String ConnectionString {
            get {
                return _stringConnection;
            }
            set {
                _stringConnection = value;
            }
        }
    }
}

```

- 6 Initialize the `private string` variable when an instance of the class is created.
- 7 Add a `public string` variable that has a method to read the value of its variable and a method to write to the `private string` variable.

- 8 Add a function that returns a string.
- 9 Add the `Get` and `Set` functions for the `public string`, which work with the `private` variable.



## Extra

You can create a function in the data layer that takes the title type as a parameter. The following code shows you how to do this. The connection string is used to construct the class.

## Example:

```
namespace DataLayer_ex {
    using System;
    using System.Data;
    using System.Data.SqlClient;
    public class DataObject {
        private string _stringConnection;
        public DataObject(String connStr){
            _stringConnection = connStr;
        }
        public DataView GetTitlesForType(string stringTitleType) {
            SqlConnection sqlconnectionPubs = new SqlConnection(_stringConnection);
            SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter
                ("select title, notes, price from " + stringTitleType + " titles where type=" +
                stringTitleType + "'", sqlconnectionPubs);
            DataSet datasetTitles = new DataSet();
            sqldataadapterTitles.Fill(datasetTitles, "Titles");
            return datasetTitles.Tables["Titles"].DefaultView;
        }
    }
}
```

```

}
public DataObject(String connStr){
    _stringConnection = connStr;
}
public String ConnectionString {
    get {
        return _stringConnection;
    }
    set {
        _stringConnection = value;
    }
}
}
public DataView GetBusinessTitles() {
    SqlConnection sqlconnectionPubs = new
    SqlConnection(_stringConnection);
    SqlDataAdapter sqldataadapterTitles = new
    SqlDataAdapter ("select title, notes, price from titles where type='business'",
    sqlconnectionPubs);
    DataSet datasetTitles = new DataSet();
    sqldataadapterTitles.Fill(datasetTitles, "Titles");
    return datasetTitles.Tables["Titles"].DefaultView;
}
}

```

**10** Create a function that returns a `DataView`.

**11** Create a `SqlConnection` that uses the `stringConnection` property.

**12** Create a `SQLDataAdapter` variable that uses the connection to get all business titles.

**13** Create a `DataSet` and fill with the data retrieved with the `SQLDataAdapter`.

**14** Return the default view from the `DataSet`.

```

C:\WINNT\System32\cmd.exe
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\>cd C:\inetpub\wwwroot
C:\inetpub\wwwroot>csc /t:library /out:bin\DataObject.dll DataObject.cs
Microsoft (R) Visual C# Compiler Version 7.00.9148 [CLR version v1.0.2615]
Copyright (C) Microsoft Corp 2000. All rights reserved.

C:\inetpub\wwwroot>

```

**15** Go to the command prompt.

**16** Compile the component using the `csc` command.

CONTINUED

# CREATE A TWO-TIER WEB FORM

The skill set for a developer that creates presentation layer code and layout is different than for a developer who writes data access code. Two-tiered Web applications allow for your Web application code to be in separate files, which makes it beneficial if you want to divide work based on developers' skill sets. Having separate files yields fewer problems with source control and allows for parallel development efforts. That is, two people can work on the same part of the site together by having separate files for presentation and data.

The Web Forms you create in ASP.NET for accessing data access components need to import the namespace for the data access component. This is done using the `@Import` directive at the top of the page. Next, you can place a Server form containing controls on the page to display the data. One way to display data is using a `DataList` control. See page 130 for more information on working with the `DataList` control. After you have specified what to show in the `DataList`, you can create the code to populate the control. You can use the `Page_Load` function to do this.

## CREATE A TWO-TIER WEB FORM

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="DataLayer" %>

<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Here are some of the resources to help you reach your business goals...

<FORM RUNAT="Server">
</FORM>

</FONT>
</BODY>
</HTML>

```

```

Untitled - Notepad
File Edit Format Help
<HEAD>

</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Here are some of the resources to help you reach your business goals...

<FORM RUNAT="Server">
<P/>
<ASP:DATALIST ID="datalistBusinessTitles" RUNAT="Server">
<ITEMTEMPLATE>
<%# DataBinder Eval(Container.DataItem, "title") %> <BR/>
<%# DataBinder Eval(Container.DataItem, "notes") %> <BR/>
<%# DataBinder Eval(Container.DataItem, "price") %> <P/>
</ITEMTEMPLATE>
</ASP:DATALIST>
</FORM>

</FONT>
</BODY>
</HTML>

```

**17** Open the `GenericTemplate.aspx` template from the Code Templates directory.

**18** Import the `DataLayer` namespace.

**19** Add a server form to the page.

**20** Add a `DataList` to the page to display the results.

**21** Add an `ItemTemplate` to the page to describe the output for each item in the result set.

## Apply It

You can create a Web page that retrieves different types of titles based on user input by putting the following code into a server-side script block. You need to compile `DataObject_ai.cs` with `DataObject_ai.bat` to have this sample run. To see the full source code for this example, see `TwoTierWebForm_ai.aspx` on the CD-ROM.

### TYPE THIS:

```
public void Page_Load(Object sender, EventArgs E) {
    if (!IsPostBack) {
        DataObject dataobjectPubs = new DataObject("server=(local)
            \\NetSDK;uid=QUser;pwd=QPassword;database=pubs");
        dataListTitles.DataSource = dataobjectPubs.GetTitlesForType("business");
        dataListTitles.DataBind();
        labelTitleType.Text = "business"; } }
public void Submit_Click(Object sender, EventArgs E) {
    DataObject dataobjectPubs = new DataObject("server=(local)\\NetSDK;
        uid=QUser;pwd=QPassword;database=pubs");
    dataListTitles.DataSource = dataobjectPubs.GetTitlesForType
        (dropdownlistTitleTypes.SelectedItem.Value.ToString());
    dataListTitles.DataBind();
    labelTitleType.Text = dropdownlistTitleTypes.SelectedItem.Text.
        ToString().ToLower();
}
```

### RESULT:

A data-bound list displays a list of books that are dependent on which title type is chosen by the user.

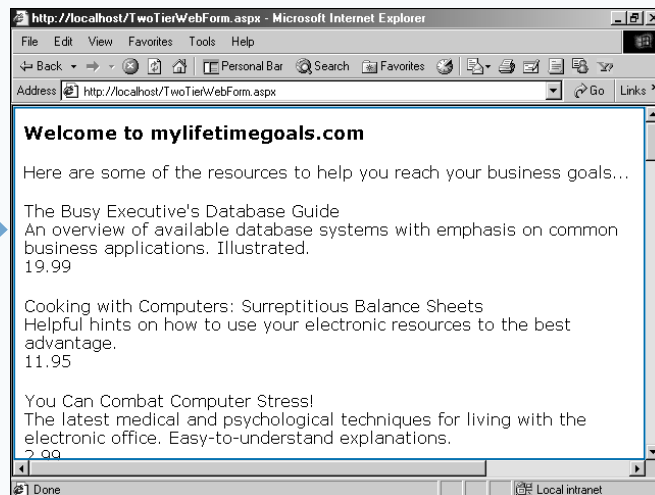
```
Untitled - Notepad
File Edit Format Help

<%@ Import Namespace="DataLayer" %>

<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
public void Page_Load(Object sender, EventArgs E) {
    DataObject dataobjectPubs = new
DataObject("server=(local)\\NetSDK;uid=QUser;pwd=QPassword;database=pubs");
    dataListBusinessTitles.DataSource =
dataobjectPubs.GetBusinessTitles();
    dataListBusinessTitles.DataBind();
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Here are some of the resources to help you reach your business goals...

<FORM RUNAT="Server">
<P>
<ASP:DATALIST ID="datalistBusinessTitles" RUNAT="Server">
</ITEMTEMPLATE>
```



**22** Add the `Page_Load` function to the page.

**23** Create a new instance of the `DataObject` class and set the connection string.

**24** Set the `DataSource` of the `DataList` on the Web page to the function in the data access Layer that returns the business titles.

**25** Bind the `DataList`.

**26** Save the file and request it from the Web server.

The title, notes, and price appear for the business titles from the pubs database using the data layer.

# CREATE A THREE-TIER WEB FORM

A business tier enables you to encapsulate business rules or logic into components. These business tier components enable you to automate business processes that your company uses. With Web applications, you can break your code out into three tiers—Data, Business, and Presentation. This allows you to consolidate all data access code into one component, all of the code related to business logic into another component, and all of the code for the user interface in the Web page.

Like the Data Access Layer, the Business Layer will be implemented as a C# component. In this component, you can set the connection string when the class is

created. You then need to add a function that will call the Data Access Layer to retrieve the data necessary to apply the business logic. After all the code necessary for mimicking your business process is put in place, you need to compile the business component to the `/bin` directory as a library.

Using a business component on a Web page is the same as using a data access component. Now that you have put a Business Layer between your Data Access Layer and your Web Page, you can change the Presentation Layer without having to rewrite business logic or data access code.

## CREATE A BUSINESS LAYER

```

Untitled - Notepad
File Edit Format Help
namespace BusinessLayer {
    using System;
    using System.Data;
    using System.Data.SqlClient;
    using DataLayer;
    public class BusinessObject {
    }
}

```

**1** Open a new document in your text editor.

**2** Create a new namespace.

**3** Add an alias to the `System`, `System.Data`, `System.Data.SqlClient`, and the `DataLayer` namespaces.

**4** Create a public class.

```

Untitled - Notepad
File Edit Format Help
namespace BusinessLayer {
    using System;
    using System.Data;
    using System.Data.SqlClient;
    using DataLayer;
    public class BusinessObject {
        private DataObject dataobjectPubs;
        public BusinessObject() {
            dataobjectPubs = new
            DataObj("server=(local)\NetSDK;uid=QSUser;pwd=QSPassword;database=pubs");
        }
    }
}

```

**5** Create a private `DataObject` class from the `DataLayer` namespace.

**6** Add the code to initialize the `BusinessObject` connection string.

## Apply It

You can extend the Business Layer by adding the `GetTitlesForType` function. This Business Layer uses the Extra Data Access Layer.

### TYPE THIS:

```
namespace BusinessLayer_ai {
    using System;
    using System.Data;
    using System.Data.SqlClient;
    using DataLayer_ex;
    public class BusinessObject {
        private DataObject dataobjectPubs;
        public BusinessObject() {
            dataobjectPubs = new
DataObject ("server=(local)\NetSDK;uid=QSUser;pwd=QSPassword;database=pubs");
        }
        public DataView GetBusinessTitles() {
            return dataobjectPubs.GetBusinessTitles();
        }
        public DataView GetTitlesForType(string stringTitleType) {
            return dataobjectPubs.GetBusinessTitles(stringTitleType);
        }
    }
}
```

### RESULT:

A business layer that is between the user interface layer and the data access layer.

```
namespace BusinessLayer {
    using System;
    using System.Data;
    using System.Data.SqlClient;
    using DataLayer;
    public class BusinessObject {
        private DataObject dataobjectPubs;
        public BusinessObject() {
            dataobjectPubs = new
DataObject("server=(local)\NetSDK;uid=QSUser;pwd=QSPassword;database=pubs");
        }
        public DataView GetBusinessTitles() {
            return dataobjectPubs.GetBusinessTitles();
        }
    }
}
```

**7** Add the function that calls `GetBusinessTitles` from the Data Layer.

```
C:\WINNT\System32\cmd.exe
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\>cd C:\inetpub\wwwroot
C:\inetpub\wwwroot> /r:bin\DataObject.dll /t:library /out:bin\BusinessObject.dll BusinessObject.cs
Microsoft (R) Visual C# Compiler Version 7.00.9148 [CLR version v1.0.2615]
Copyright (C) Microsoft Corp 2000. All rights reserved.

C:\inetpub\wwwroot>
```

**8** Go to the command prompt.

**9** Compile the component using the `csc` command.

CONTINUED

# CREATE A THREE-TIER WEB FORM

Implementing a Business Layer into a tiered application allows for clean separation between business logic and the user interface. This gives you the flexibility of having multiple user interfaces for your application without having to rewrite business and data access code.

The business layer is a critical part of your application and is where you need to truly understand the rules that enforce good business practices. Sometimes these practices need to ensure that you interact with the database without compromising data integrity. This can be done with transactional code. The business layer is traditionally the location for where transactional code is placed to ensure integrity of your data stores.

To write code that uses your business component, you need to import the namespace for the business component. This is done using the `@Import` directive at the top of the page. If you are pulling data back from your business component, you can place a Server form on the page with a control to display the data. One way to display data is using a `DataList` control. See page 130 for more information on working with the `DataList` control. After you have specified what to show in the `DataList`, you can then create the code to populate the control. You can use the `Page_Load` function to do this.

## CREATE A THREE-TIER WEB FORM

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="BusinessLayer" %>
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Here are some of the resources to help you reach your business goals...

<FORM RUNAT="Server">
</FORM>

</FONT>
</BODY>
</HTML>
  
```

**10** Open the `GenericTemplate.aspx` template from the Code Templates directory.

**11** Import the `BusinessLayer` namespace.

**12** Add a server form to the page.

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="BusinessLayer" %>
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Here are some of the resources to help you reach your business goals...

<FORM RUNAT="Server">
</FORM>
<P/>
<ASP:DATALIST ID="datalistBusinessTitles" RUNAT="Server">
<ITEMTEMPLATE>
<%# DataBinder.Eval(Container.DataItem, "title") %> <BR/>
<%# DataBinder.Eval(Container.DataItem, "notes") %> <BR/>
<%# DataBinder.Eval(Container.DataItem, "price") %> <P/>
</ITEMTEMPLATE>
</ASP:DATALIST>
</FONT>
</BODY>
</HTML>
  
```

**13** Add a `DataList` to the page to display the results.

**14** Add an `ItemTemplate` to the page to describe the output for each item in the result set.

## Apply It

You can use a business layer to control access to data. To run this sample, you need to compile `DataObject_ai.cs` and `BusinessObject_ai.cs` to the `/bin` directory as a library (using the `DataObject_ai.bat` and `BusinessObject_ai.bat` batch files, respectively). Then place the following code into a server-side script block. To see the full source code for this example, see `ThreeTierWebForm_ai.aspx` on the CD-ROM.

### TYPE THIS:

```
public void Page_Load(Object sender, EventArgs E) {
    if (!IsPostBack) {
        BusinessObject businessobjectPubs = new
            BusinessObject("server=(local)\\NetSDK;uid=QSUser; " +
                "pwd=QSPassword;database=pubs");
        datalistBusinessTitles.DataSource = businessobjectPubs.GetBusinessTitles();
        datalistBusinessTitles.DataBind();
        labelTitleType.Text = "business"; } }

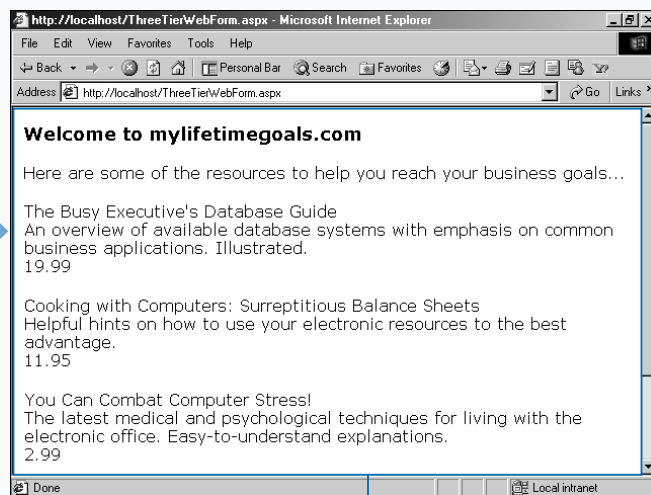
public void Submit_Click(Object sender, EventArgs E) {
    BusinessObject businessobjectPubs = new
        BusinessObject("server=(local)\\NetSDK;uid=QSUser; " +
            "pwd=QSPassword;database=pubs");
    datalistTitles.DataSource =
        businessobjectPubs.GetTitlesForType
        (dropdownlistTitleTypes.SelectedItem.Value.ToString());
    datalistTitles.DataBind();
    labelTitleType.Text = dropdownlistTitleTypes.SelectedItem.Text.ToString().ToLower;
}
```

### RESULT:

A data bound list that displays a list of books that are dependent on which title type is chosen by the user.

```
Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="BusinessLayer" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
public void Page_Load(Object sender, EventArgs E) {
    BusinessObject businessobjectPubs = new BusinessObject();
    datalistBusinessTitles.DataSource =
businessobjectPubs.GetBusinessTitles();
    datalistBusinessTitles.DataBind();
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">
<H3>Welcome to mylifetimagoals.com</H3>
Here are some of the resources to help you reach your business goals...

<FORM RUNAT="Server">
<P>
<ASP:DATALIST ID="datalistBusinessTitles" RUNAT="Server">
<ITEMTEMPLATE>
<%# DataBinder.Eval(Container.DataItem, "title") %> <BR/>
<%# DataBinder.Eval(Container.DataItem, "note") %> <BR/>
```



**15** Add the `Page_Load` function to the page.

**16** Create a new instance of the `DataObject` class and set the connection string.

**17** Set the `DataSource` of the `DataList` on the Web page to the function in the Data Access Layer that returns the business titles.

**18** Bind the `DataList`.

**19** Save the file and request it from the Web server.

The title, notes, and price appear for the business titles from the pubs database using the business and data layers.

# USE A CODE-BEHIND FOR YOUR ASP.NET PAGE

**Y**ou can store your code in a separate file, called a Code-behind, which allows for you to clearly separate the code from presentation. This enables people with Web design skills to work on pages separately from the Web programmers. This is a big advantage in the ASP.NET framework and was not available in ASP 3.0. In ASP 3.0, your server-side code had to be inline with your HTML (on the same page).

To utilize a Code-behind in your ASP.NET page, you add two attributes to the page directive. One of the attributes is the `Inherits` attribute, which you can

use to specify the class which you want to use in your Code-behind. The next attribute, the `Src` attribute, specifies the location of the file that contains the Code-behind code. In the Code-behind file, you can implement event handlers as if they were on the page that uses the Code-behind. For example, a common event handler is the `Page_Load` event.

All the code in the Code-behind is server-side code that may require round trips to the Web server. To avoid this, you need to use client-side code in the Web form.

## USE A CODE-BEHIND FOR YOUR ASP.NET PAGE

```

Untitled - Notepad
File Edit Format Help
using BusinessLayer;
using System;
using System.Web.UI;
using System.Web.UI.WebControls;

public class CodeBehind : Page {

    public DataList dataListBusinessTitles;

}
  
```

```

Untitled - Notepad
File Edit Format Help
<%@ Page Inherits="CodeBehind" Src="CodeBehind.cs" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
public void Page_Load(Object sender, EventArgs E) {
    BusinessObject businessobjectPubs = new BusinessObject();
    dataListBusinessTitles.DataSource =
businessobjectPubs.GetBusinessTitles();
    dataListBusinessTitles.DataBind();
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Here are some of the resources to help you reach your business goals...

<FORM RUNAT="Server">
<P/>
<ASP.DATALIST ID="dataListBusinessTitles" RUNAT="Server">
<ITEMTEMPLATE>
<%# DataBinder.Eval(Container.DataItem, "title") %> <BR/>
  
```

- 1 Open your text editor.
- 2 Create an alias to the `BusinessLayer`, `System`, `System.Web.UI`, and `System.Web.UI.WebControls`.
- 3 Create a public class of type `Page`.
- 4 Create a public variable of type `DataList`.

- 5 Open the `ThreeTierWebForm.aspx` template from the Code Templates directory.
- 6 Add a `Page` directive at the top of the page and an `Inherits` attribute with a value of the class name and a `Src` attribute with the value of the filename.
- 7 Cut the `Page_Load` event handler from the file.
- 8 Save the file as the class name to the Default Web site.



**Apply It**

You can use an event handler fired from a control as well. This code example calls the `Button_OnClick` event handler on the Code-behind page. The first section of code is to associate the Code-behind page with the `aspx` page, this goes at the top of the `aspx` page. The second section of code goes into the Code-behind page.

**TYPE THIS AT THE TOP OF THE WEB FORM:**

```
<%@ Page Inherits="CodeBehind_ai" Src="CodeBehind_ai.cs" %>
```

**TYPE THIS IN THE CODE-BEHIND PAGE:**

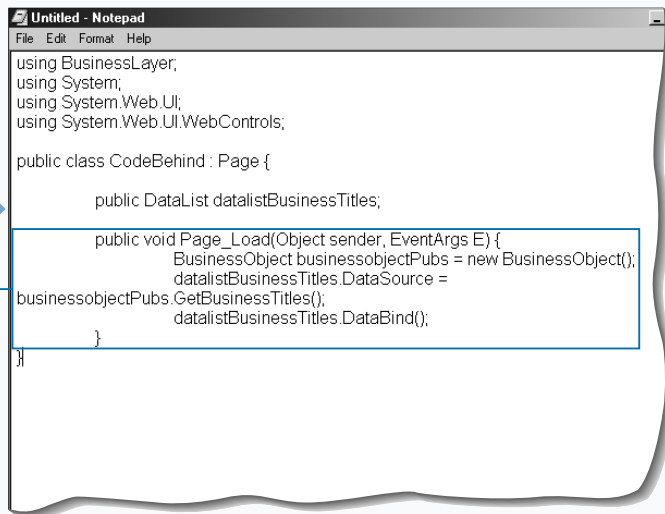
```
using BusinessLayer;
using System;
using System.Web.UI;
using System.Web.UI.WebControls;

public class CodeBehind_ai : Page {
    public Label labelMessage;

    public void Button_OnClick(object Source, EventArgs e) {
        labelMessage.Text="You are going to Step 2...";
    }
}
```

**RESULT:**

The Web page calls the Code-behind when the button is clicked.



**9** Paste the `Page_Load` event handler into the CodeBehind page.

**10** Save the file and request it from the Web server.

The title, notes, and price appear for the business titles from the pubs database using the business and data layers in a Code-behind.

# READ FORM DATA WITH REQUEST.FORM

The `HttpRequest` object enables you to read the HTTP values sent by a user during a Web request. This way of working with user input is how you typically access user input with ASP 3.0, giving you a backward compatibility. There is no true backward compatibility using C# as the language; but if you are using VB, it is compatible.

You can use `Request.Form` and `Request.QueryString` methods to read data that is submitted from another Web page. In order to specify which control you want to access, you need to know the control's name or ID. For example, you can use the drop-down list box in the task that has an ID of `dropdownlistSuggestions`. To read the

value of this control, you can use `Request.Form["dropdownlistSuggestions"]`.

The configuration of your HTML form determines if you use the `Form` or `QueryString` to retrieve form data. The HTML form has a `METHOD` attribute. If this attribute is set to `POST`, then you need to use `Request.Form` to obtain user input from the HTML form. If set to `METHOD="GET"`, then you need to use `Request.QueryString`. Both of the properties on the `Request` object (`Form` and `QueryString`) contain a `NameValueCollection` collection class. The `NameValueCollection` class represents a sorted collection of associated `String` keys and `String` values that can be accessed either with the key or index.

## READ FORM DATA WITH REQUEST.FORM

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
What suggestions do you have for improving our site?

<FORM RUNAT="Server">
</FORM>

</FONT>
</BODY>
</HTML>

```

**1** Open the `GenericTemplate.aspx` template from the Code Templates directory.

**2** Add a heading for the page.

**3** Add a message to the user.

**4** Add a form to the page.

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
What suggestions do you have for improving our site?

<FORM RUNAT="Server">
<P/>
<ASP.DROPDOWNLIST ID="dropdownlistSuggestions" RUNAT="Server">
  <ASP.LISTITEM>Fewer Goals</ASP.LISTITEM>
  <ASP.LISTITEM>More Goals</ASP.LISTITEM>
  <ASP.LISTITEM>Same Number of Goals</ASP.LISTITEM>
</ASP.DROPDOWNLIST>
<ASP.BUTTON ID="buttonExample" RUNAT="Server" TEXT="Submit"/><P/>
<ASP.LABEL ID="labelButtonExample" RUNAT="Server"/>
</FORM>

</FONT>

```

**5** Place a drop-down list box on the page with suggestions for choosing the suggested number of goals.

**6** Add a button control to the page.

**7** Add a label control to the page.

## Apply It

You can pass data in a URL by appending name-value pairs to the end of the address. Note the METHOD="GET" for the form variable.

### TYPE THIS:

```
<HTML>
<HEAD><SCRIPT LANGUAGE="C#" RUNAT="Server">
public void Page_Load(Object sender, EventArgs E) {
    if (IsPostBack) {
        labelButtonExample.Text =
Request.QueryString["dropdownlistSuggestions"].ToString();
    }
}
</SCRIPT></HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
What suggestions do you have for improving our site?
<FORM RUNAT="Server" METHOD="GET">
<P/>
<ASP:DROPDOWNLIST ID="dropdownlistSuggestions" RUNAT="Server">
    <ASP:LISTITEM>Fewer Goals</ASP:LISTITEM>
    <ASP:LISTITEM>More Goals</ASP:LISTITEM>
    <ASP:LISTITEM>Same Number of Goals</ASP:LISTITEM>
</ASP:DROPDOWNLIST>
<ASP:BUTTON ID="buttonExample" RUNAT="Server" TEXT="Submit" /><P/>
<ASP:LABEL ID="labelButtonExample" RUNAT="Server" />
</FORM>
</FONT></BODY></HTML>
```

### RESULT:

A Web page that, when a selection is made and the form is submitted, places the form values into the query string.



```

<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
public void Page_Load(Object sender, EventArgs E) {
    if (IsPostBack) {
        labelButtonExample.Text =
Request.Form["dropdownlistSuggestions"].ToString();
    }
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
What suggestions do you have for improving our site?

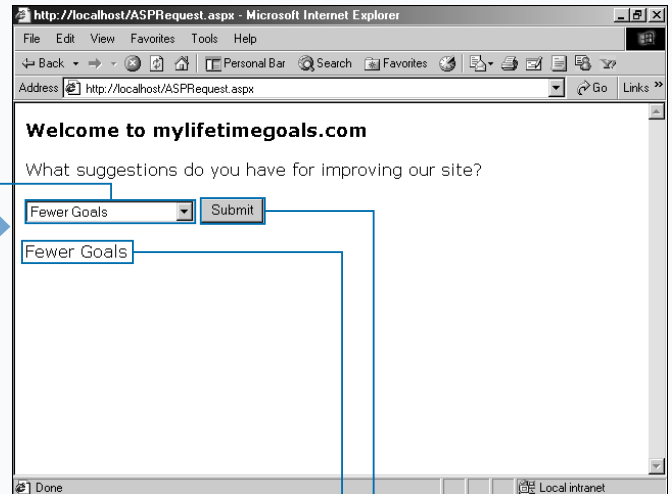
<FORM RUNAT="Server">
<P/>
<ASP:DROPDOWNLIST ID="dropdownlistSuggestions" RUNAT="Server">
    <ASP:LISTITEM>Fewer Goals</ASP:LISTITEM>
    <ASP:LISTITEM>More Goals</ASP:LISTITEM>
    <ASP:LISTITEM>Same Number of Goals</ASP:LISTITEM>
</ASP:DROPDOWNLIST>
<ASP:BUTTON ID="buttonExample" RUNAT="Server" TEXT="Submit" />
</FORM>
</FONT>
</BODY>
</HTML>

```

**8** Add the `Page_Load` function.

**9** Add an `if` statement to make sure that the code will only run when posting to the page.

**10** Add the `Request.Form` method to retrieve the value that was selected in the drop-down list box.



**11** Save the file and request it from the Web server.

**12** Click  and select a suggestion.

**13** Click the Submit button.

The suggestion you chose appears.

# DISPLAY DATA WITH REQUEST.PARAMS

You can use the `Request` object's `Params` method to obtain a combined collection of `QueryString`, `Form`, `ServerVariables`, and `Cookies` items. This will give you most of the data that is in a Web request. Before ASP.NET, this combined collection of the `QueryString`, `Form`, `ServerVariables`, and `Cookies` items was the default collection of the `Request` object. This information is still available directly from the `Request` object, but to be more explicit, you can access through `Params`.

You can pull request information from any page that requests a Web page and allows server-side code. Therefore, a simple HTML page can be the source for the `Request.Params` collection. For example, you

can pull form data from an HTML form on a simple html page. To do this, you will need to have the HTML form post to the Web page that will process the HTTP request. On this processing page you can use the `Request.Params` collection to obtain any form data.

If you are not sure of the requesting page's form method, then the `Params` collection is very useful. The method you use determines if the form data is available in the `Form` collection (`METHOD="POST"`) or the `QueryString` collection (`METHOD="GET"`). Because the `Params` collection has both of these collections combined, you can just pull the value from `Params`.

## DISPLAY DATA WITH REQUEST.PARAMS

```

Unlitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
What suggestions do you have for improving our site?

<FORM ACTION="ASPRequestParams2.aspx">
<P/>
<SELECT ID="selectSuggestions">
  <OPTION>Fewer Goals</OPTION>
  <OPTION>More Goals</OPTION>
  <OPTION>Same Number of Goals</OPTION>
</SELECT>
<INPUT TYPE="Submit" VALUE="Submit">
<P/>
</FORM>

</FONT>
</BODY>
</HTML>

```

**1** Open the `Suggestions.htm` template from the Code Templates directory.

**2** Add the `ACTION` value to the form.

The contents of the page contain a suggestions form.

**3** Save the file.

```

Unlitled - Notepad
File Edit Format Help
<% @ PAGE LANGUAGE="C#" %>
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Here is all the data from Request.Params.<P/>

<% GetRequestParams();%>

</FORM>
</FONT>
</BODY>
</HTML>

```

**4** Open the `GenericTemplate.aspx` template from the Code Templates directory.

**5** Add a heading to the file.

**6** Add a message to the file.

**7** Add the script delimiters.

**8** Call the `GetRequestParams` function.

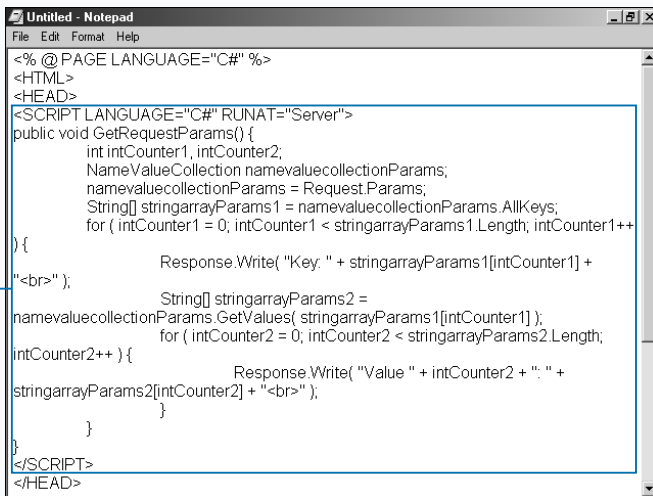
**Extra**

You can just display the form variables using the following code:

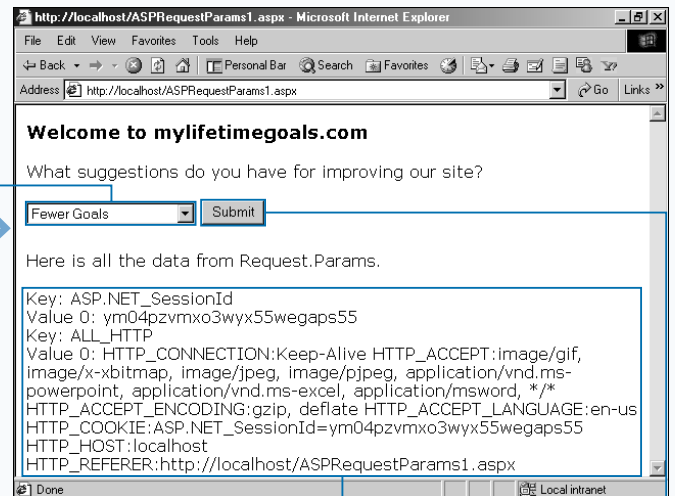
**Example:**

```
<SCRIPT LANGUAGE="C#" RUNAT="Server">
public void GetRequestForm() {
    int intCounter1;
    NameValueCollection namevaluecollectionForm;
    namevaluecollectionForm=Request.Form;
    String[] stringarrayForm1 = namevaluecollectionForm.AllKeys;
    for (intCounter1 = 0; intCounter1 < stringarrayForm1.Length; intCounter1++) {
        Response.Write("Form: " + stringarrayForm1[intCounter1] + "<BR/>");
    }
}
</SCRIPT>

<% GetRequestForm(); %>
</FORM>
</FONT>
</BODY>
</HTML>
```



**9** Add the `GetRequestParams` function and code from the file `GetRequestParams.aspx` located in the Code Templates directory.



**10** Save the file and request it from the Web server.

**11** Request the first Web page that you created in this task.

**12** Click  and select a suggestion.

**13** Click the Submit button.

Request data displays from looping through the `Request.Params` collection.

# WRITE OUTPUT USING RESPONSE.WRITE

You can use the `HttpResponse` class to interact with the responses given to Web requests. One common task for the `HttpResponse` is to write custom HTML in the response to a user's page request. Writing custom HTML can be accomplished with the `Write` method of the `Response` object. Note that the methods and properties of the `HttpResponse` class are exposed through ASP.NET's intrinsic `Response` object.

To get user input to display in a new HTML element, you can programmatically add HTML elements/tags with `Response.Write`. When using the

`Response.Write` to customize the HTML markup of a page, you need to know where you want to have the tag(s) placed. You can place in a placeholder tag in which you can insert HTML SPAN or DIV tags), or you can use the script delimiters inline to the HTML.

If you are posting a page back to itself, you can check the `IsPostBack` property of the page to determine if it is the first time you are displaying the page or if you have posted back to the same page. In some cases, you will only want to run certain server-side code after a page is posted back to itself.

## WRITE OUTPUT USING RESPONSE.WRITE

```

Untitled - Notepad
File Edit Format Help
<% @ PAGE LANGUAGE="C#" %>
<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
What suggestions do you have for improving our site?

<FORM RUNAT="Server">
</FORM>

</FONT>
</BODY>
</HTML>

```

**1** Open the `GenericTemplate.aspx` template from the Code Templates directory.

**2** Add a heading for the page.

**3** Add a message to the user.

**4** Add a form to the page.

```

Untitled - Notepad
File Edit Format Help
<% @ PAGE LANGUAGE="C#" %>
<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
What suggestions do you have for improving our site?

<FORM RUNAT="Server">
<P/>
<ASP.DROPDOWNLIST ID="dropdownlistSuggestions" RUNAT="Server">
  <ASP.LISTITEM>Fewer Goals</ASP.LISTITEM>
  <ASP.LISTITEM>More Goals</ASP.LISTITEM>
  <ASP.LISTITEM>Same Number of Goals</ASP.LISTITEM>
</ASP.DROPDOWNLIST>
<ASP.BUTTON ID="buttonExample" RUNAT="Server" TEXT="Submit"/><P/>
</FORM>

</FONT>
</BODY>

```

**5** Place a drop-down list box on the page with suggestions for choosing the suggested number of goals.

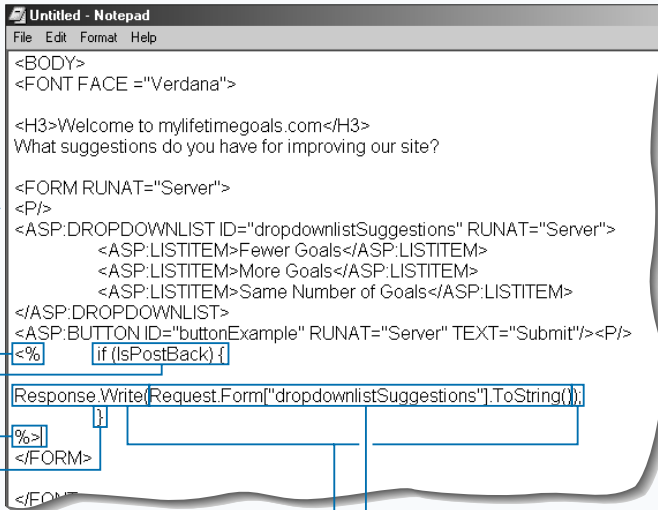
**6** Add a button control to the page.

**Extra**

If you were passing this information via the query string, the code would look a little different compared to posting through a form. See `ASPResponse_ai.aspx` for the full example on the CD.

**Example:**

```
<% @ PAGE LANGUAGE="C#" %>
<H3>Welcome to mylifetimegoals.com</H3>
What suggestions do you have for improving our site?
<FORM RUNAT="Server" METHOD="GET">
<P/>
<ASP:DROPDOWNLIST ID="dropdownlistSuggestions" RUNAT="Server">
  <ASP:LISTITEM>Fewer Goals</ASP:LISTITEM>
  <ASP:LISTITEM>More Goals</ASP:LISTITEM>
  <ASP:LISTITEM>Same Number of Goals</ASP:LISTITEM>
</ASP:DROPDOWNLIST>
<ASP:BUTTON ID="buttonExample" RUNAT="Server" TEXT="Submit" /><P/>
<%
  if (IsPostBack) {
    Response.Write(Request.QueryString["dropdownlistSuggestions"]);
  }
%>
</FORM>
```

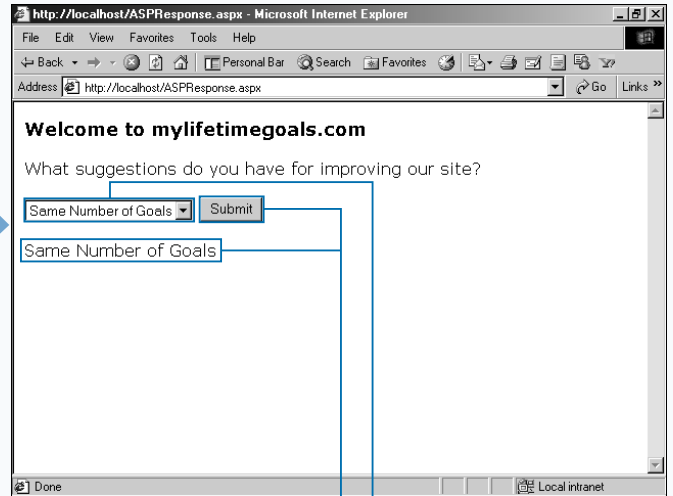


**7** Add a set of script delimiters after the button.

**8** Add an `if` statement to make sure that the code will only run when posting to the page.

**9** Add the `Request.Form` method to retrieve the value that was selected in the drop-down list box.

**10** Add the `Response.Write` method to write the value selected in the drop-down list box.



**11** Save the file and request it in

■ The Web page for submitting suggestions appears.

**12** Click  and select a suggestion.

**13** Click the Submit button.  
■ The suggestion you chose appears.

# REDIRECT USING RESPONSE.REDIRECT

You can use the `HttpResponse` class to redirect users to other pages besides the page they originally requested. One situation where you can use a redirect is when you delete an existing page off of your Web site. For example, if you come up with a new naming convention for pages on your site, you can keep the old pages on the site with a redirect to the replacement page. You can also use a redirect to handle an error on a Web page. When the error occurs, you can redirect them to a standard error page. Another common use of redirects is on a page

that processes the user's request and redirects the user based on what is in the user's `Request` object.

To redirect a user's request, you can use `Response.Redirect`. When performing redirects, you need to make sure that no HTTP response packets have been sent to the user requesting the page. If any HTTP packets have been sent and you perform a redirect, you will generate a server error. To avoid getting this server error, you can buffer the response by adding the page directive at the top of the file and setting `Buffer=True`.

## REDIRECT USING RESPONSE.REDIRECT

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
What suggestions do you have for improving our site?

<FORM METHOD="POST" ACTION="ASPResponseRedirect2.aspx">
<P/>
<SELECT NAME="selectSuggestions">
  <OPTION>Fewer Goals</OPTION>
  <OPTION>More Goals</OPTION>
  <OPTION>Same Number of Goals</OPTION>
</SELECT>
<INPUT TYPE="Submit" VALUE="Submit">
<P/>
</FORM>

```

- 1 Open the `Suggestions.htm` template from the Code Templates directory.
- 2 Add the `ACTION` value to the form.

- 3 The contents of the page contains a suggestions form.
- 3 Save the file.

```

Untitled - Notepad
File Edit Format Help
<% @ PAGE LANGUAGE="C#" Buffer="True" %>

<%
string stringSuggestion = Request.Form["selectSuggestions"].ToString();

%>

```

- 4 Open a new document in your text editor.
- 5 Add the page directive, set the language to `C#`, and buffer the page.
- 6 Add a pair of script delimiters.
- 7 Request the data from the drop-down list box and read it into a `string` variable.



# Apply It

You can create a page that accepts a page name from the `QueryString` and redirects the user to that page.

### TYPE THIS:

```
<% @ PAGE LANGUAGE="C#" Buffer="True"%>

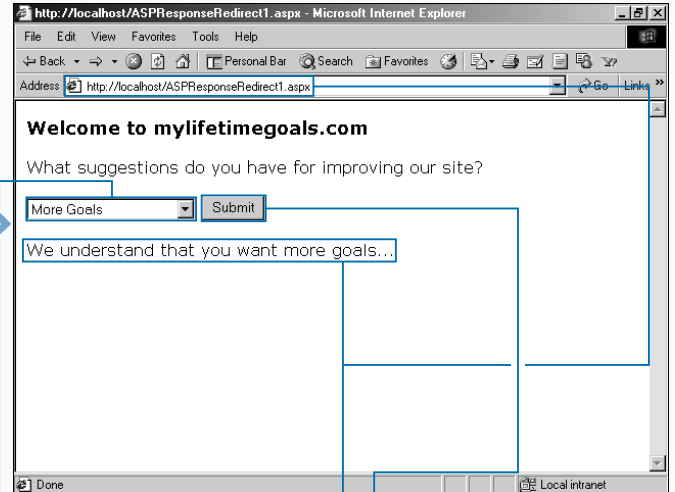
<%
if (Request.QueryString["pageName"] != null) {
    string stringPageName = Request.QueryString["pageName"]
        .ToString();
    Response.Redirect(stringPageName);
}
else {
    Response.Redirect(Request.Url.ToString() +
        "?pageName=ie.aspx");
}
%>
```

### RESULT:

A request to this page with a query string equal to `pageName=request.aspx` will redirect the request to the `request.aspx` page. If the page is not specified, then you are redirected to `ie.aspx`.

```
Untitled - Notepad
File Edit Format Help
<% @ PAGE LANGUAGE="C#" Buffer="True" %>

<%
string stringSuggestion = Request.Form["selectSuggestions"].ToString();
switch(stringSuggestion) {
case "Fewer Goals":
    Response.Redirect("FewerGoals.aspx");
    break;
case "More Goals":
    Response.Redirect("MoreGoals.aspx");
    break;
case "Same Number of Goals":
    Response.Redirect("SameGoals.aspx");
    break;
default:
    Response.Redirect("ErrorGoals.aspx");
    break;
}
%>
```



- 8 Create a `switch` statement using the `string` variable.
- 9 Add a case for each of the options that redirects to the appropriate page.
- 10 Save the file and request from the Web server.

- 11 Copy the files `FewerGoals.aspx`, `MoreGoals.aspx`, `SameGoals.aspx`, and `ErrorGoals.aspx` from the CD-ROM to your working directory.

- 12 Click  and select a suggestion.
- 13 Click the Submit button. You are redirected to a page according to the option you have selected.

# CHECK FOR WEB BROWSER TYPES

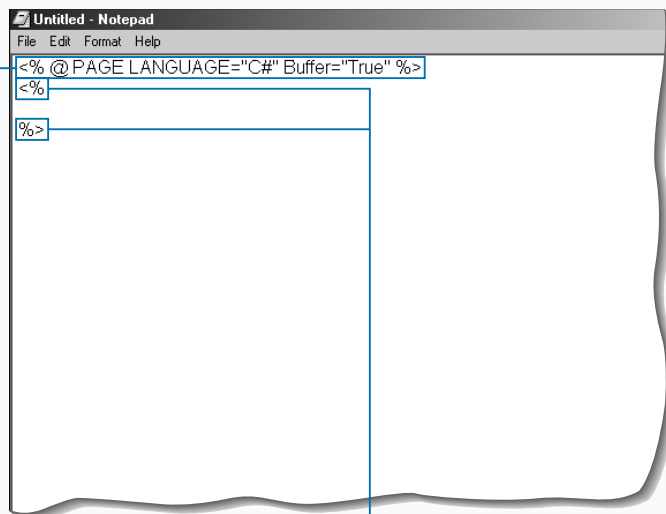
You can use the `HttpBrowserCapabilities` class to find out the properties of a user's Web browser. You can then use this information in your code to determine what the proper response to your client should be. Perhaps you might redirect the user to another page based on the browser type. You could also use the `HttpBrowserCapabilities` class for information to do custom logging that tracks what types of browsers are accessing your site.

Some sites that you build with depend on browser capabilities for making decisions on what is sent to the user of the site. For example, you might have a few pages on your site that can be enhanced with ActiveX controls. Before sending the ActiveX control

in the response, you want to check to see whether the users support ActiveX controls. If they do not, you can redirect them to a page that is implemented without an ActiveX control.

To use the `HttpBrowserCapabilities` class, you need to create a variable of type `HttpBrowserCapabilities`. With this variable, you can use the `Request.Browser` property to return all of the information about the user's Web browser. After you have this information, you can use the property of interest. For example, the `Browser` property is used to determine where to send the user.

## CHECK FOR WEB BROWSER TYPES



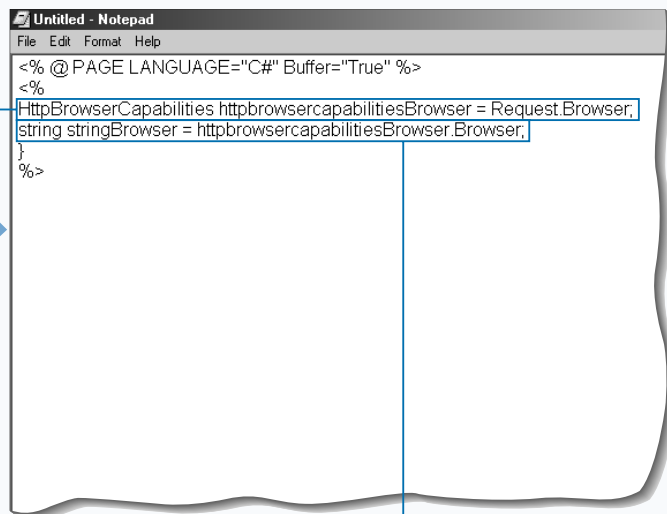
```

Untitled - Notepad
File Edit Format Help
<% @ PAGE LANGUAGE="C#" Buffer="True" %>
<%
%>
  
```

**1** Open a new document in your text editor.

**2** Add the page directive, set the language for the page to C#, and buffer the page.

**3** Add a pair of script delimiters.



```

Untitled - Notepad
File Edit Format Help
<% @ PAGE LANGUAGE="C#" Buffer="True" %>
<%
HttpBrowserCapabilities httpbrowsercapabilitiesBrowser = Request.Browser;
string stringBrowser = httpbrowsercapabilitiesBrowser.Browser;
%>
  
```

**4** Create a new variable of type `HttpBrowserCapabilities` and initialize the variable by using the `Request.Browser` property.

**5** Create a new variable of type `string` and read the Web browser type property into the string.

**Apply It**

The following code is a sampling of all the information that you can collect about a Web browser by using the `HttpBrowserCapabilities` object. See `ASPBrowsersCapabilities_ai.aspx` on the CD for the full example.

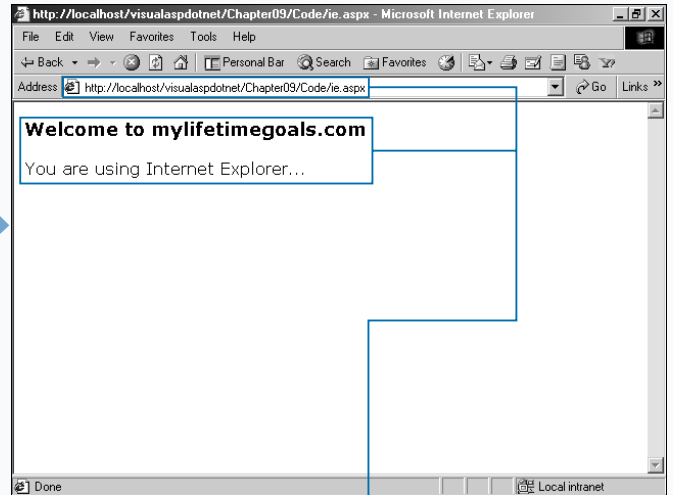
**TYPE THIS:**

```
<H3><%=Request.Url.ToString()%></H3>
Here is all the data from Request.Browser.<P/>
<%
HttpBrowserCapabilities hbcBrowser = Request.Browser;
Response.Write( "Type = " + hbcBrowser.Type + "<BR/>" );
Response.Write( "Name = " + hbcBrowser.Browser + "<BR/>" );
Response.Write( "Version = " + hbcBrowser.Version + "<BR/>" );
Response.Write( "Supports Frames = "
+ hbcBrowser.Frames + "<BR/>" );
Response.Write( "Supports Tables = "
+ hbcBrowser.Tables + "<BR/>" );
Response.Write( "Supports Cookies = "
+ hbcBrowser.Cookies + "<BR/>" );
Response.Write( "Supports ActiveX Controls = "
+ hbcBrowser.ActiveXControls + "<BR/>" );
%>
</FORM>
```

**RESULT:**

```
http://localhost:81/new/ASP
BrowserCapabilities_ai.
aspx
Here is all the data from
Request.Browser.
Type = IE6
Name = IE
Version = 6.0b
Supports Frames = True
Supports Tables = True
Supports Cookies = True
Supports ActiveX Controls =
True
```

```
Untitled - Notepad
File Edit Format Help
<% @ PAGE LANGUAGE="C#" Buffer="True" %>
<%
HttpBrowserCapabilities httpbrowsercapabilitiesBrowser = Request.Browser;
string stringBrowser = httpbrowsercapabilitiesBrowser.Browser;
switch(stringBrowser) {
case "IE":
    Response.Redirect("IE.aspx");
    break;
default:
    Response.Redirect("Other.aspx");
    break;
}
%>
```



- 6** Create a `case` statement using the `string` variable.
- 7** Add a `case` for each of the options that redirects to the appropriate page.

- 8** Save the file.
- 9** Copy `IE.aspx` and `Other.aspx` from the CD-ROM to your working directory.

- 10** Request the file from the Web server.

- You are redirected to a page according to your Web browser type.

# SEND AN E-MAIL USING ASP.NET

You can send e-mail from your ASP.NET Web pages with the `System.Web.Mail` namespace to respond to a request that a user makes on your site. E-mail gives you a convenient means to send users feedback, such as receipts, confirmation notes, and other information that a user likes to have for future reference. You can also use e-mail to work with other systems; for example, an encrypted e-mail containing order information can be sent from your Web site to another system for processing.

To send e-mail from an ASP.NET Web page, first import the `System.Web.Mail` namespace. In this namespace, you can work with a couple of objects.

The most important is the `MailMessage` object that has many of the properties that you need to send an e-mail. You can set the message `From`, `To`, `Subject`, `Body`, and `BodyFormat` properties when creating the e-mail. To send the prepared mail message, you need to work with the `SMTPMail` object.

To send the e-mail message, you need an SMTP server. SMTP mail service is built into Microsoft Windows 2000. To ensure that your Web server is not blocked, mail is queued by default on a Windows 2000 system.

## SEND AN E-MAIL USING ASP.NET

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Web.Mail" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void SubmitBtn_Click(object Source, EventArgs e) {
    string stringMessage = "";
    stringMessage = "<FONT FACE='Verdana'>My suggestion is for " +
dropdownlistSuggestions.SelectedItem.Text + "</FONT>";
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">
<H3>Welcome to mylifetimagoals.com</H3>
What suggestions do you have for improving our site?

<FORM RUNAT="Server">
<P/>
<ASP:DROPDOWNLIST ID="dropdownlistSuggestions" RUNAT="Server">
  <ASP:LISTITEM>Fewer Goals</ASP:LISTITEM>
  <ASP:LISTITEM>More Goals</ASP:LISTITEM>
  <ASP:LISTITEM>Same Number of Goals</ASP:LISTITEM>
</ASP:DROPDOWNLIST>

```

**1** Open the `SuggestionsTemplate` template from the Code Templates directory.

**2** Import the `System.Web.Mail` namespace.

**3** In the `SubmitBtn_Click` function, create a `string` variable for creating your message.

**4** Set the message contents in HTML, using the selected value from the drop-down list box.

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Web.Mail" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void SubmitBtn_Click(object Source, EventArgs e) {
    string stringMessage = "";
    stringMessage = "<FONT FACE='Verdana'>My suggestion is for " +
dropdownlistSuggestions.SelectedItem.Text + "</FONT>";
    MailMessage mailmessageSuggestion = new MailMessage();
    mailmessageSuggestion.From = "user@mylifetimagoals.com";
    mailmessageSuggestion.To = "president@mylifetimagoals.com";
    mailmessageSuggestion.Subject = "Suggestion";
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">
<H3>Welcome to mylifetimagoals.com</H3>
What suggestions do you have for improving our site?

<FORM RUNAT="Server">
<P/>
<ASP:DROPDOWNLIST ID="dropdownlistSuggestions" RUNAT="Server">

```

**5** Create a new variable of `MailMessage` type.

**6** Set the `From` property for the `MailMessage`.

**7** Set the `To` property for the `MailMessage`.

**8** Set the `Subject` property for the `MailMessage`.

**Extra**

The e-mail format can either be HTML format or Text format. Because some e-mail clients will not format HTML files, you may wish to send e-mail in Text format. Many sites allow the user to specify whether they wish to receive e-mail in HTML or Text format.

You can let the user select the priority of the e-mail message with a drop-down list box. See `ASPMail_ai.aspx` on the CD for an example of using this in a full `aspx` page.

**Example:**

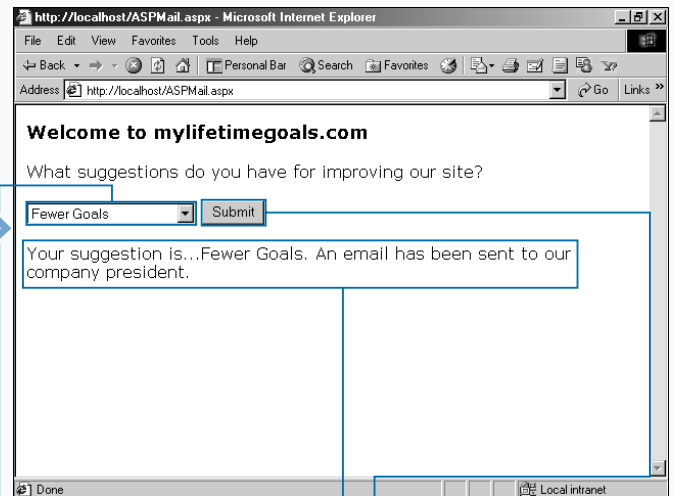
```
String stringMailPriority =
    dropdownlistMailPriority.SelectedItem.Text;

switch(stringMailPriority) {
    case "High":
        mailmessageSuggestion.Priority = MailPriority.High;
        break;
    case "Normal":
        mailmessageSuggestion.Priority = MailPriority.Normal;
        break;
    case "Low":
        mailmessageSuggestion.Priority = MailPriority.Low;
        break;
    default:
        mailmessageSuggestion.Priority = MailPriority.Normal;
        break;
}
```

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Web.Mail" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void SubmitBtn_Click(object Source, EventArgs e) {
    string stringMessage = "";
    stringMessage = "<FONT FACE='Verdana'>My suggestion is for " +
    dropdownlistSuggestions.SelectedItem.Text + "</FONT>";
    MailMessage mailmessageSuggestion = new MailMessage();
    mailmessageSuggestion.From = "user@mylifetimegoals.com";
    mailmessageSuggestion.To = "president@mylifetimegoals.com";
    mailmessageSuggestion.Subject = "Suggestion";
    mailmessageSuggestion.Body = stringMessage;
    mailmessageSuggestion.BodyFormat = MailFormat.Html;
    SmtpMail.Send(mailmessageSuggestion);
    labelMessage.Text = "Your suggestion is..." +
    dropdownlistSuggestions.SelectedItem.Text + ". An email has been sent to our company
    president.";
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">

```



**9** Set the `Body` property for the `MailMessage`, using the `string` variable in which you have placed the message.

**10** Set the `BodyFormat` property for the `MailMessage` to `MailFormat.Html`.

**11** Add the `SmtpMail`'s `Send` method to send the message.

**12** Update the `Text` property for the label on the page to reflect that the mail has been sent.

**13** Save the file and request from the Web server.

**14** Click  and select a suggestion.

**15** Click the Submit button.

The message appears, notifying you of the e-mail's status.

# USE THE ASP.NET PAGE CACHE

One way of increasing performance for your ASP.NET Web pages is to cache pages on the Web server. When using a page cache, ASP.NET does not generate a new response for a Web page every time it is requested. This performance optimization can be used on pages that are accessed frequently and or have little to no personalization on them.

The process of setting up page caching is simple. On the page you wish to cache, add an `OutputCache` directive at the top of the page and add a `Duration` attribute for the directive. The `Duration` will specify how long to cache the page in seconds. There will be

cases when you may want a page generated based on the request made. For example, you may want to have another version of the page generated when something varies in the request's query string.

When caching pages, ASP.NET is smart about sending cached pages. For example, if a page is requested with a query string that is different from the cached page, then ASP.NET will regenerate the page and cache this new page (keeping the original page cached). The next time the page is requested, it will check to see whether that version of the page is cached before reprocessing the page.

## USE THE ASP.NET PAGE CACHE

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
protected void Page_Load(Object sender, EventArgs e) {
    SqlConnection sqlconnectionPubs = new
SqlConnection("server=(local)\NetSDK;uid=QSUser,pwd=QSPasswrd,databas
s");
    SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter("select t
notes, price from titles where type='business'", sqlconnectionPubs);
    DataSet datasetTitles = new DataSet();
    sqldataadapterTitles.Fill(datasetTitles, "titles");
    datagridTitles.DataSource=datasetTitles.Tables["titles"].DefaultView;
    datagridTitles.DataBind();
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3>Welcome to multifati...
  
```

**1** Open the `DatagridTemplate.aspx` template from the Code Templates directory.

```

Untitled - Notepad
File Edit Format Help
<%@ OutputCache Duration="60" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
protected void Page_Load(Object sender, EventArgs e) {
    SqlConnection sqlconnectionPubs = new
SqlConnection("server=(local)\NetSDK;uid=QSUser,pwd=QSPasswrd,databas
s");
    SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter("select t
notes, price from titles where type='business'", sqlconnectionPubs);
    DataSet datasetTitles = new DataSet();
    sqldataadapterTitles.Fill(datasetTitles, "titles");
    datagridTitles.DataSource=datasetTitles.Tables["titles"].DefaultView;
    datagridTitles.DataBind();
    labelMessage.Text = "Generated on:" + DateTime.Now.ToString("G");
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
  
```

**2** Add the `OutputCache` directive to set the duration of the page cache to 60 seconds.

**3** Add a message to the page by updating the label with the current time.

*Note: The page-generation message will update only when the code in the `Page_Load` function is run.*

**4** Save the file and request it from the Web server.

**Extra**

If you want to cache the page per user session, you can cache based on the users cookie. Authentication cookies are used to map a user to a session. Open two instances of Internet Explorer and note the generated times for each instance and note they are not sharing the same cached page. See `ASPPageCache_ex1.aspx` on the CD for a page that demonstrates this directive.

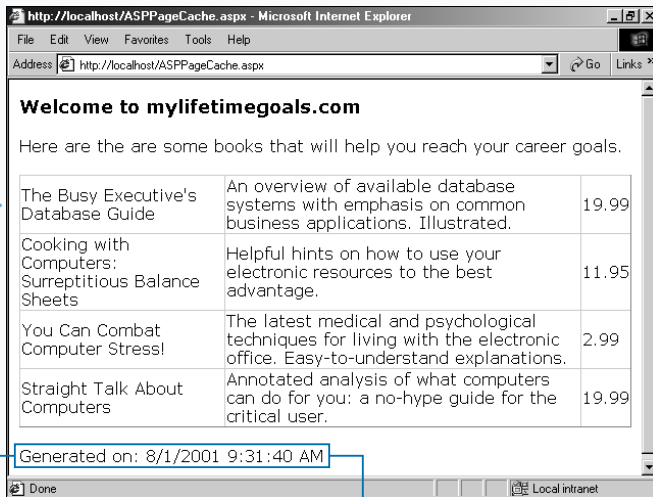
**Example:**

```
<%@ OutputCache Duration="60"
VaryByHeader="Cookie" %>
```

You can bypass a cached page each time the user submits different `Form` or `QueryString` data to a page. See `ASPPageCache_ex2.aspx` on the CD for a page that demonstrates this directive.

**Example:**

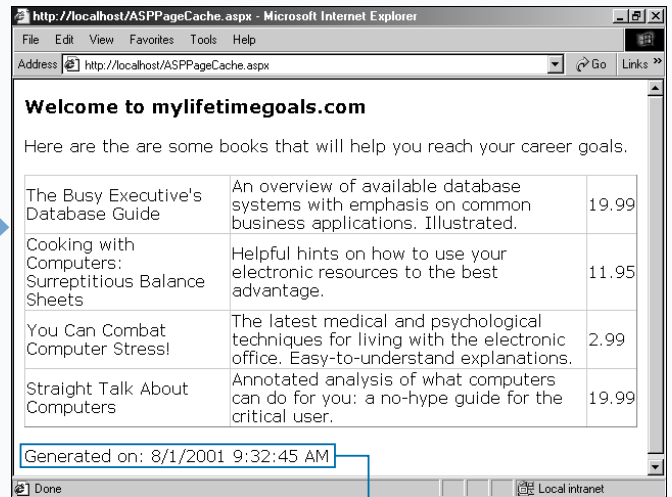
```
<%@ OutputCache Duration="60" VaryByParam="*" %>
```



You can observe that the generated time is 9:31:40 AM.

**5** Press F5 to refresh the page.

You can observe the generated time is the same as before (9:31:40 AM), indicating you are viewing a cached page.



**6** Wait for more than 60 seconds and refresh the Web page.

The generated time updates, indicating that the cached page was not used.

# USE THE ASP.NET DATA CACHE

You can increase performance on your ASP.NET Web pages by placing data for your Web pages in a cache on the Web server. When using a data cache, you can cache a dataset on the Web server so that you do not have to go back to the database when the page is refreshed.

To place data into the cache, you can use `Cache["name"] = value`, where `name` is the name you want to access the information by in code and `value` is the value for the name. To read the data out of the cache, you can use `variable = Cache["name"]`. Note that for storing data into

cache, you cast the data in the cache to a `DataRowView` data type. With the cache, you can set memory and duration by respectively setting the length of the cache and setting a sliding expiration for the cache.

When caching data on your Web pages, you need to check to see if the cache exists before using it. If it does not exist—for instance, it expires—then you execute the code necessary to retrieve the data from the database again. One way of checking to see if the cache exists is by checking to see if the cache is `null`.

## USE THE ASP.NET DATA CACHE

```

<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
protected void Page_Load(Object sender, EventArgs e) {
    DataView dataviewTitles;
    dataviewTitles = (DataView)Cache["cacheTitles"];
    if (dataviewTitles == null) {
        SqlConnection sqlconnectionPubs = new
        SqlConnection("server=(local)\NetSDK;uid=QsUser;pwd=QSPassWord;database=p
        s");

        SqlDataAdapter sqldataadapterTitles = new
        SqlDataAdapter("select title, notes, price from titles where type='business'",
        sqlconnectionPubs);

        DataSet datasetTitles = new DataSet();
        sqldataadapterTitles.Fill(datasetTitles, "titles");

        datagridTitles.DataSource=datasetTitles.Tables["titles"].DefaultView,
        datagridTitles.DataBind();
    }
}
</SCRIPT>

```

**1** Open the `DatagridTemplate.aspx` template from the Code Templates directory.

**2** Create a new `DataView` variable.

**3** Try to read the `DataView` out of the page cache.

**4** If there was nothing in the cache to retrieve, get a new `DataView`.

```

protected void Page_Load(Object sender, EventArgs e) {
    DataView dataviewTitles;
    dataviewTitles = (DataView)Cache["cacheTitles"];
    if (dataviewTitles == null) {
        SqlConnection sqlconnectionPubs = new
        SqlConnection("server=(local)\NetSDK;uid=QsUser;pwd=QSPassWord;database=p
        s");

        SqlDataAdapter sqldataadapterTitles = new
        SqlDataAdapter("select title, notes, price from titles where type='business'",
        sqlconnectionPubs);

        DataSet datasetTitles = new DataSet();
        sqldataadapterTitles.Fill(datasetTitles, "titles");
        dataviewTitles = new DataView(datasetTitles.Tables["titles"]);
        Cache["cacheTitles"] = dataviewTitles;
        labelMessage.Text = "Dataset created explicitly.";
    }
    else {
        labelMessage.Text = "Dataset retrieved from cache.";
    }
    datagridTitles.DataSource=dataviewTitles;
    datagridTitles.DataBind();
}

```

**5** Create the new `DataView`.

**6** Cache the `DataView`.

**7** Update the label on the page to show that the `DataSet` was created explicitly.

**8** Add an `else` message to indicate that the data cache was used.

**9** Update the `DataSource` to be `dataviewTitles`.



**Extra**

You can specify how long the data is in the cache. The following code will expire the data cache after one minute.

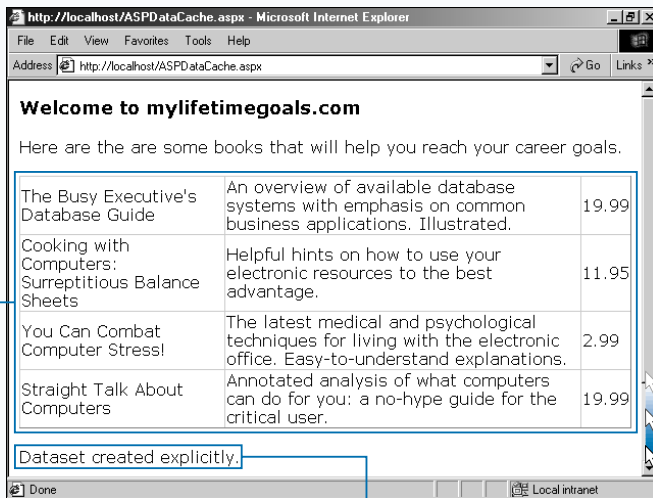
**Example:**

```
Cache.Insert("cacheTitles", dataviewTitles,
    null,DateTime.Now.AddMinutes(1), TimeSpan.Zero );
```

You can base expiration on the last time the cache was accessed by using a sliding expiration.

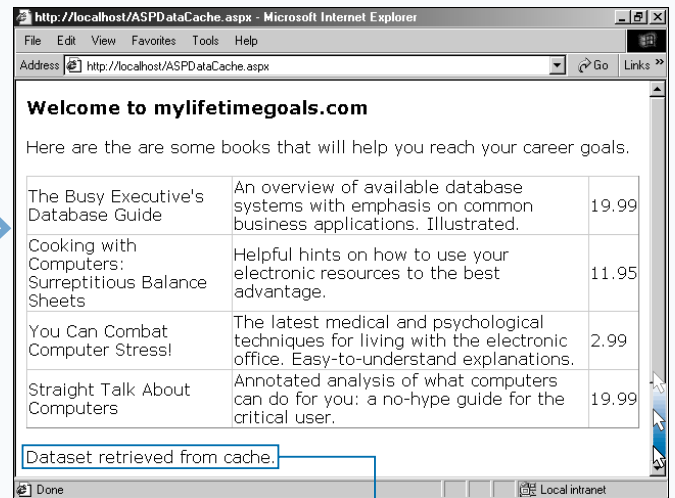
**Example:**

```
Cache.Insert("cacheTitles", dataviewTitles,
    null,DateTime.MaxValue, TimeSpan.FromMinutes(1) );
25
```



**10** Save the file and request it from the Web server.  
 ■ The business titles from the pubs database appear.

**11** Scroll down to see the message about the DataSet being created explicitly.



**12** Press F5 to refresh the page.

**13** Scroll down to see the message about the DataSet being retrieved from the cache.

# INTRODUCTION TO ASP.NET APPLICATIONS AND STATE MANAGEMENT

## ASP.NET APPLICATIONS

An ASP.NET Application is how you can separate one application domain from another. An ASP.NET Application is set up as a Web site or a virtual directory that has been configured as an application in Internet Information Server (IIS). All resources under the Web site or virtual directory are part of the application, unless the resource is in a subdirectory that has been configured as another application.

Within an ASP.NET Application, you can independently control security, process isolation, and shared data. The Web sites you build can include multiple Web applications as part of the Web site, or one main application for your entire site.

Having one or more Web applications on your Web site depends on several factors. Some of the more important factors that you should be aware of are:

### Application Protection

One benefit of separate Web applications is the control of where your Web application's process runs. This process is the management of threads that are allocated to running processing instructions for your application. Process Isolation is determined by the *Application Protection* setting in the IIS MMC (Microsoft Management Console for IIS). You have three options to choose from on a Web site for what process isolation you need. The levels of *Application Protection* are:

#### Low (IIS Process)

This provides the best performance but is the least stable. This means that if your Web application crashes, the whole Web server comes down.

#### Medium (Pooled)

Runs in a central process where all "pooled" applications run. This is second in performance compared to low isolation and will not bring down IIS when it crashes, but will bring down all other pooled applications.

#### High (Isolated)

Runs in its own process. This is the most stable option, but its performance is not as good as low isolation (due to interprocess communication between the Web application and IIS).

### Security Model

All applications have several security options that can be set across the site, directory level, or even down to file level through the IIS MMC's Directory/File Security tab in the Properties dialog box. Depending on which option you choose for Application Protection, you can have more or less control on security for your entire application. The setting that you choose for Application Protection will determine which COM+ Application will contain your application. With low and medium protection levels, you will have to share the COM+ Application with other Web applications. In the case of low protection, you will share the COM+ Application with IIS. With the high protection level, your application will have its own COM+ Application.

### Shared Information

Each ASP.NET application will have its own set of resources to manage global data in your application. When you start storing state into global locations, you exhaust server resources (primarily memory). See the section on application state management on page 211 for global storage options with Application and Session objects. If you find that you are storing large amounts of data in global storage and this data can be separated into logical independent domains, then it is time to segment into one or more separate ASP.NET applications.

After your Web site or virtual directory is set up as an application, add a simple file with an `.aspx` extension to it. When you request this file through your Web server, the .NET runtime creates the ASP.NET application context.

## WORKING WITH AND EXTENDING ASP.NET APPLICATIONS

**Global.asax**

Each ASP.NET Application you create will only use one `Global.asax` file. The Application will look for this file in the root directory of the site.

The `Global.asax` file contains code for certain events that are used during the lifetime of your ASP.NET application, including code that executes when the application starts and finishes, as well as code for custom events. With the `Global.asax` file, you can also configure settings for your application, including declaring namespaces and server-side objects for use throughout the application.

**Server-side Objects**

You can declare server-side objects in the `Global.asax` file for use in your application. You can specify the scope of the object based upon how often your application uses the object. Most of the time, you will declare server-side objects at the page level. You can also specify the scope to be at the application or session level. Be careful with this scope, because you will be keeping the object in memory for an infinite amount of time, causing inefficient use of memory.

**Application Directives**

Application directives give you the ability to specify additional information about your application. You can work with three application directives in ASP.NET: Application, Import, and Assembly.

**Custom Modules**

You can extend the `Global.asax` file to execute more than just the standard events. You can create your own event-handling code for when specific events occur in your application.

## APPLICATION STATE MANAGEMENT

**HTTP Application**

When ASP.NET creates your application, it will create multiple instances of an application object. You can write code to overwrite what happens by default when each of these application objects are created or destroyed.

**Session Object**

You may wish to have variables that are specific to a user for the life of their visit to your application. For this, you can use the `Session` object. You can initialize the `Session` variables in the `Session_Start` event handle in the `Global.asax` file. These variables can then be easily read and updated for each individual user.

**Application Object**

You may wish to have certain variables available from any page on your site, for any user, and at any time. These variables can be any `Common Type System` data type, including a server-side object. You will use the `Application` object to store these variables for the life of the application.

**Cookies**

*Cookies* enable you to store information on the user's browser. A cookie can be either a single piece of information or an entire collection. After you store the cookie on the user's browser, you can later access the cookie and take appropriate action.

# CREATE A GLOBAL.ASAX FILE

You can use the `Global.asax` file to create special subroutines, called *event handlers*, which are executed by the Web server when specific events occur. An application can only have one `Global.asax` file. The purpose of the `Global.asax` file in ASP.NET is similar to its purpose in ASP 3.0. You should place this file in the root directory of the application. The extension of `.asax` tells the Web server that it is a Global Application file.

The `Application_Start` event fires when a user accesses the first `.aspx` page after you do one of the following things: a) reboot the Web server, b) stop and restart the World Wide Web service, c) update the `global.asax` file, or d) deploy the application

for the first time. Getting the first `.aspx` page served requires a little extra time because the code in the `Application_Start` event is being executed. You can use this event to set variables using the `Application` object, to write application information to a log file, or to do anything else that is required when the application starts.

The `Application_End` event is fired when the Web server is being shut down. You can use this event to clean up variables created in the `Application_Start` event, to log any additional Application information, or to execute any other functionality required when shutting down the application.

## CREATE A GLOBAL.ASAX FILE

```

<SCRIPT LANGUAGE="C#" RUNAT="Server">
</SCRIPT>

```

**1** Start your text editor to create a `Global.asax` file.

**2** Type `<SCRIPT LANGUAGE="C#" RUNAT="Server">` and press Enter twice.

**3** Type `</SCRIPT>`.

```

<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Application_Start() {
    string stringSiteName = "My Lifetime Goals";
    Application["applicationSiteName"] = stringSiteName;
}
</SCRIPT>

```

**4** To create the event handler for when the application starts, click between the `<SCRIPT>` and `</SCRIPT>` tags, type `void Application_Start() {`, and press Enter.

**5** Press Tab to indent, type the code you want to execute when the application starts, and press Enter.

*Note: This code will create an Application variable.*

*Note: Indent the code so that it is easier to read.*

**6** Type `}` to finish the event handler and press Enter.

**Extra**

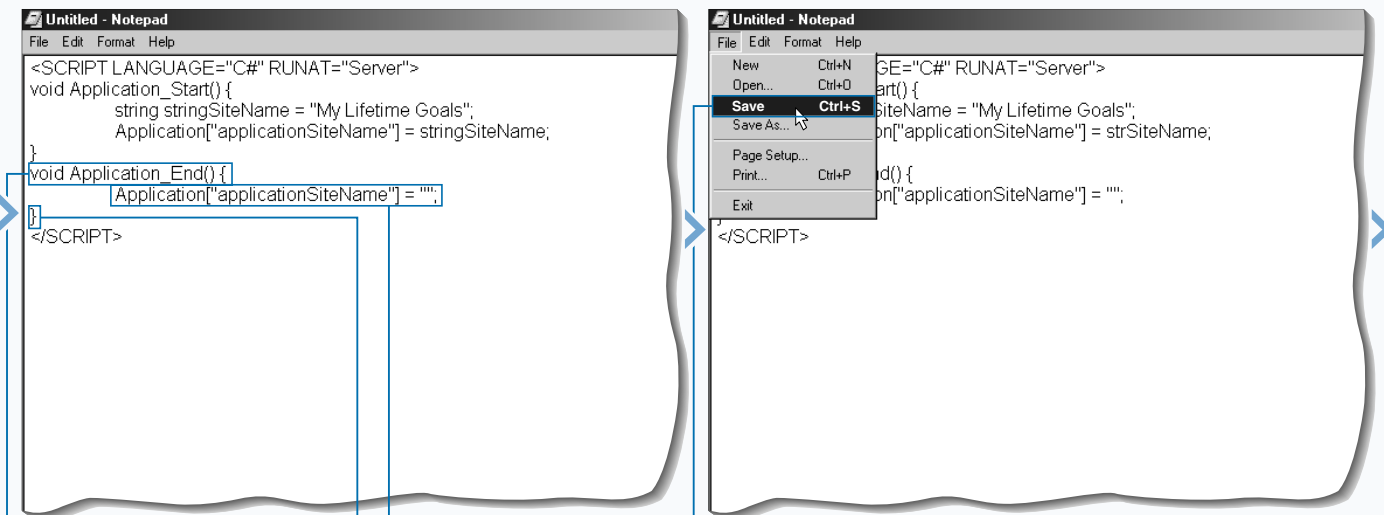
You can specify the language you want to use in the `<SCRIPT>` tag by setting the language attribute. This could include C# or VB. You will always want to specify the `RUNAT` attribute to run at the server because that is where the `Application` and `Session` objects are located. Here is what the `<SCRIPT>` tag would look like when using C# as the language:

```
<SCRIPT LANGUAGE="C#" RUNAT="Server">
```

The `Global.asax` file is an optional file. When ASP.NET finds no `Global.asax` file, the assumption is that there is no need to code for the application and session events.

The `Global.asax` file is configured so that you cannot request it as a URL. You should set up permissions on the file so that unauthorized users cannot read or update the file, especially if you have any sensitive configuration information in the file.

You do not need to stop the Web server when updating the `Global.asax` file. When you save changes to the `Global.asax` file, ASP.NET automatically detects that you have changed the file. ASP.NET then restarts the application.



**7** To create the event handler for when the application ends, click between the `<SCRIPT>` and `</SCRIPT>` tags, type `void Application_End() {`, and press Enter.

**8** Press Tab to indent, type the code you want to execute when the application ends, and press Enter.

**9** Type `}` to finish the event handler and press Enter.

**10** Click File ⇨ Save to open the dialog box.

CONTINUED

# CREATE A GLOBAL.ASAX FILE

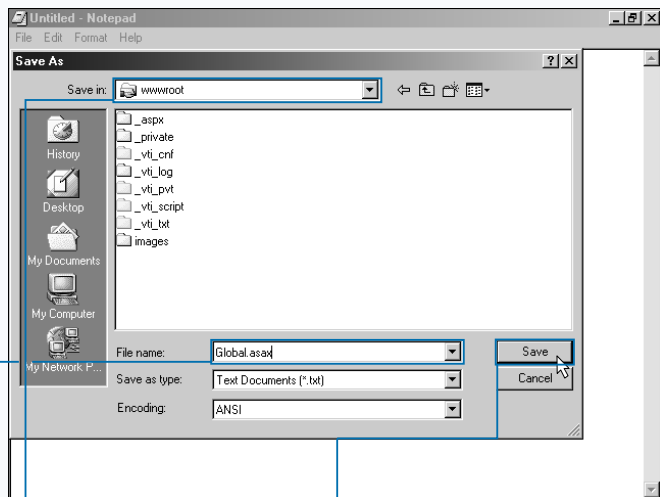
You can insert event handlers to indicate when a user's session starts and ends. For more information about sessions and the `Session` object, see page 226.

The `Session_Start` event fires when a new user accesses his or her first ASP page from your ASP.NET application. ASP.NET takes care of the details of determining whether the user is new. When ASP.NET determines that the user is new, the code in the `Session_Start` event is executed for the user. This code is run before any of the code on the ASP page that was requested. You can use this event for many

purposes, including setting variables for the user, redirecting the user to a login page, or using any other code that you want to run when a user first shows up to your Web site.

The `Session_End` event fires when the user's session times out, which happens when the time between requesting pages is greater than the time-out period set for the application, or when the session is abandoned programmatically. You can use this event to clean up variables created in the `Session_Start` event or execute any other cleanup code required when users leave your site.

## CREATE A GLOBAL.ASAX FILE (CONTINUED)



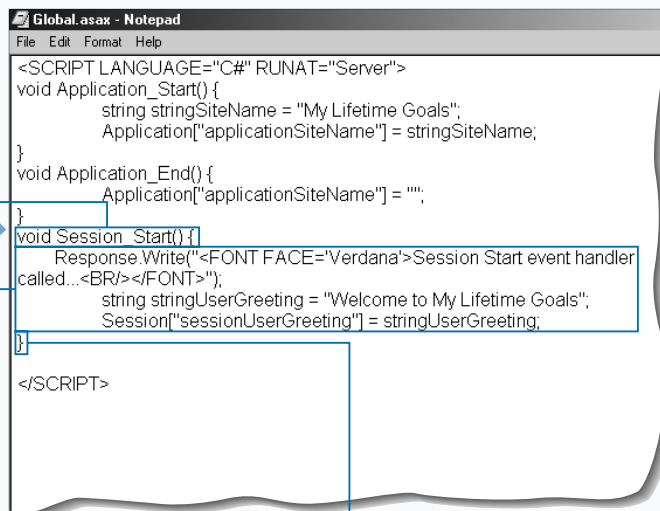
**11** Click to select the folder where you want to store your file.

**12** Type a name for the Web page.

You can save the file to the `C:\Inetpub\wwwroot\` directory with the filename of `Global.aspx`.

**13** Click Save to save the Web page.

The Default Web site now has a `Global.aspx` file.



**14** To create the event handler for when the session starts, click between the `<SCRIPT>` and `</SCRIPT>` tags, type `void Session_Start() {`, and press Enter.

**15** Press Tab to indent, type the code you want to execute when the application starts, and press Enter.

*Note: This code will write a message when the session starts and create a session variable.*

**16** Type `}` to finish the event handler and press Enter.

**Extra**

If you want to track statistics on your site, you can log information from `Global.asax` to a file.

**Example:**

```
<% @ Import Namespace="System.IO" %>

<SCRIPT LANGUAGE="C#" RUNAT="Server">

void Application_Start() {

    FileStream filestreamLog = new FileStream("applogging.txt", FileMode.Append,
        FileAccess.Write );
    StreamWriter streamwriterLog = new StreamWriter(filestreamLog);

    streamwriterLog.WriteLine("{0}", String.Concat("The application started at "
        ,DateTime.Now));
    streamwriterLog.Close();
    streamwriterLog.Close();

}

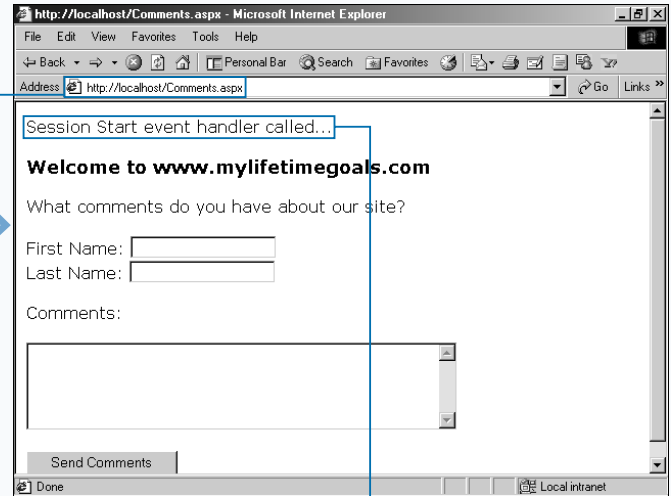
</SCRIPT>
```

```
Global.asax - Notepad
File Edit Format Help
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Application_Start() {
    string stringSiteName = "My Lifetime Goals";
    Application["applicationSiteName"] = stringSiteName;
}
void Application_End() {
    Application["applicationSiteName"] = "";
}
void Session_Start() {
    Response.Write("<FONT FACE='Verdana'>Session Start event handler
called...<BR/></FONT>");
    string stringUserGreeting = "Welcome to My Lifetime Goals";
    Session["sessionUserGreeting"] = stringUserGreeting;
}
void Session_End() {
    Session["sessionUserNickname"] = "";
}
</SCRIPT>
```

**17** To create the event handler for when the session ends, click between the `<SCRIPT>` and `</SCRIPT>` tags, type `void Session_End() {`, and press Enter.

**18** Press Tab to indent, type the code you want to execute when the application ends, and press Enter.

**19** Type `}` to finish the event handler and press Enter.



**20** Save the file.

This example is saving the `Global.asax` to the default Web site.

**21** Request an ASP.NET Web page from the default Web site.

The message about the session event handler appears.

*Note: You can request any ASP.NET Web page in the default Web site. In the example, the `Comments.aspx` ASP.NET Web page is requested.*



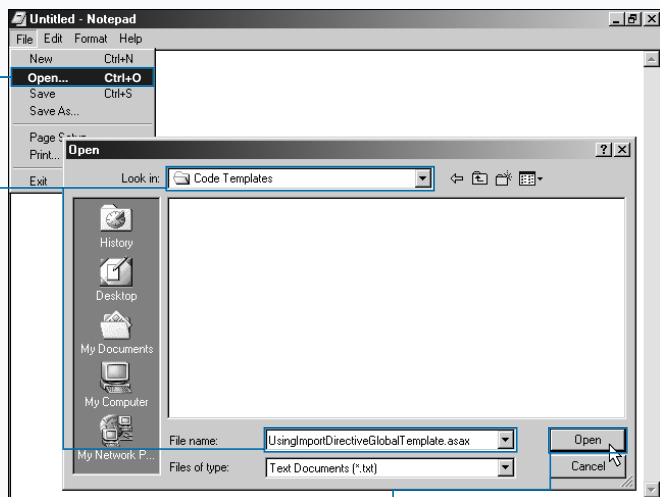
# USING PROCESSING DIRECTIVES IN THE GLOBAL.ASAX FILE

You can use processing directives in the `Global.asax` file to specify settings used in your application. The three classifications of directives are `Application`, `Assembly`, and `Import` directives. With the `Application` directive, you can set two attributes, `Inherits` and `Description`. The `Inherits` attribute is used to set which .NET base class the `global.asax` file uses for all instances of `global.asax`. The ASP.NET application compiler uses this information to compile a new application that extends the specified class. The `Description` attribute gives a short description of your application. You can place both attributes in the same directive.

The next directive that you could use is the `Assembly` directive. This directive links an assembly to the application. This makes classes in the assembly available to your application. Another directive that appears similar to the `Assembly` directive is the `Import` directive. You can use the `Import` directive to import .NET namespaces. The `Import` directive assumes that the assembly that contains the namespace is already available. After you import a namespace, you can reference classes in the namespace without giving the full qualification to the class.

As with page-level directives, you should place the application-level directives at the top of the `Global.asax` file. You should also place the directive name/value pair within the ASP.NET script delimiters.

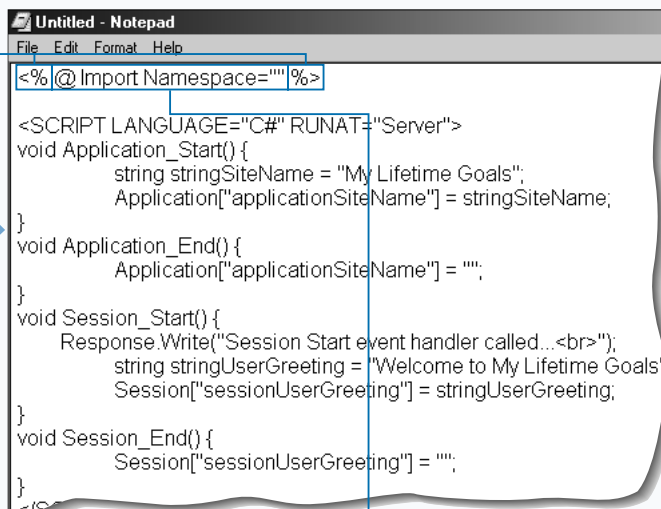
## USING THE IMPORT DIRECTIVE



- 1 Start the text editor to edit the `Global.asax` file.
- 2 Click `File` → `Open`.
- 3 Click to select the `Code Templates` directory and open the `UsingImportDirectiveGlobalTemplate.aspx` file.

- 4 Click `Open` to open the template.

*Note: You can open another `Global.asax` file that you want to use as an import directive.*



- 5 Type the ASP.NET script delimiters (`<% %>`) and press `Enter`.

- 6 Position the insertion point after the first ASP.NET script delimiter (`<%`) and type a space followed by `@ Import Namespace=""`.



**Extra**

The `Assembly` directive lets you link components to namespaces that have been imported with the `Import` directive.

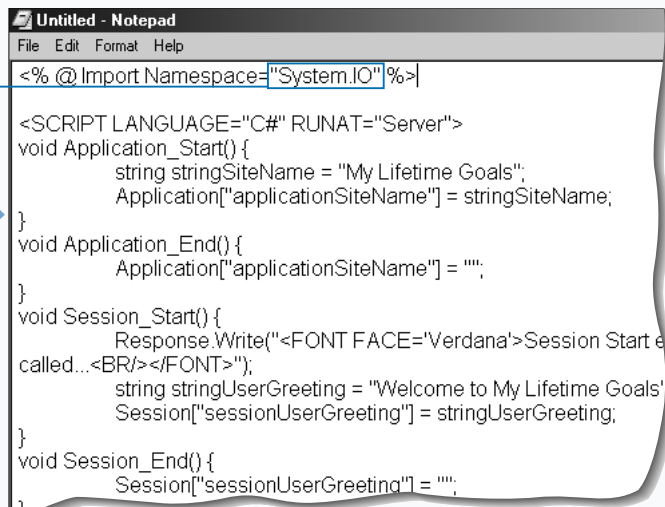
**Example:**

```
<%@ Assembly Name="MyAssembly.dll" %>
```

Another directive you can use in your `Global.asax` file is the `Application` directive. The `Inherits` attribute defines the new application base class. This will need to be a compiled .NET class on your server. You can also specify a friendly name for the application with the `Description` attribute.

**Example**

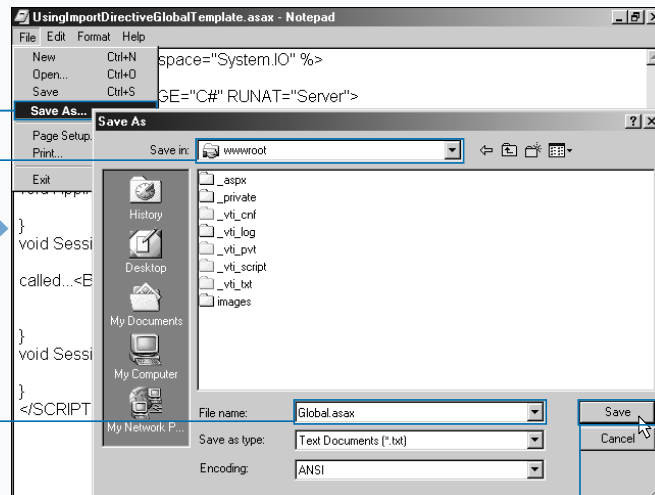
```
<%@ Application Inherits=" MySampleApp.Object"
Description="
Sample Description"%>
```



```
Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.IO"%>

<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Application_Start() {
    string stringSiteName = "My Lifetime Goals";
    Application["applicationSiteName"] = stringSiteName;
}
void Application_End() {
    Application["applicationSiteName"] = "";
}
void Session_Start() {
    Response.Write("<FONT FACE='Verdana'>Session Start e
called...<BR/></FONT>");
    string stringUserGreeting = "Welcome to My Lifetime Goals";
    Session["sessionUserGreeting"] = stringUserGreeting;
}
void Session_End() {
    Session["sessionUserGreeting"] = "";
}
```

**7** Position the insertion point between the quotation marks (" ") and type the namespace you want to import.



```
UsingImportDirectiveGlobalTemplate.asax - Notepad
File Edit Format Help
New Ctrl+N
Open... Ctrl+O
Save Ctrl+S
Save As...
Save As
Page Setup...
Print...
Save in: wwwroot
Exit
History
Desktop
My Documents
My Computer
My Network P...
File name: Global.asax
Save as type: Text Documents (*.txt)
Encoding: ANSI
Save
Cancel
```

**8** Click File ⇨ Save As to open the dialog box.

**9** Click [ ] to select the folder where you want to store your file.

**10** Type a name for the Web page.

**11** Click Save to save the Web page.

*Note: Click the Yes button if you are prompted about replacing the file.*

■ The namespace is now declared for all pages on the site.

# USING SERVER-SIDE OBJECTS IN THE GLOBAL.ASAX FILE

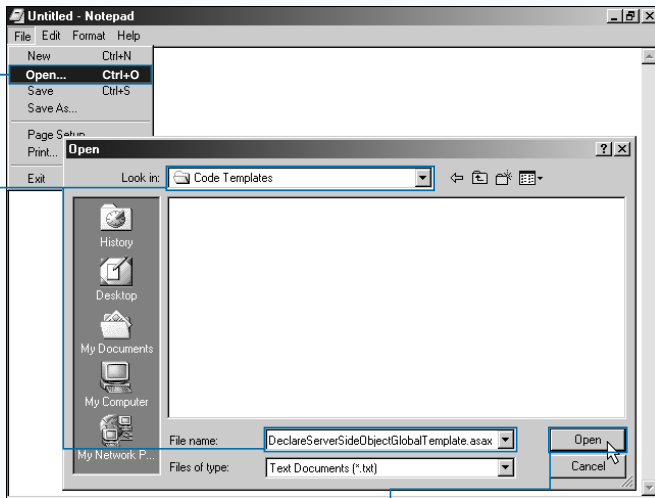
You can use the `<OBJECT>` tag in the `Global.asax` file to declare an object for use throughout your ASP.NET application. You should place the `<OBJECT>` tag outside the code declaration blocks. Therefore, you can place the `<OBJECT>` tag before or after the `<SCRIPT>` tags, but not before any processing directives (`<%@ Directive %>`). For more about processing directives, see page 216.

The scope of the server-side objects are either for the entire application, per application instance, or per user session. You can set this using the `scope` attribute. When the `scope` attribute is set to `application`, there is a single object that is

available for all users, on every page, at any time. When the scope is set to `appinstance`, there are multiple objects. There is one for each application instance, which is available to all users, on every page, at any time. When set to `session`, each user session manages a unique copy of the object.

Because of the scope of these variables, you should use caution when deciding what you place into memory. Use `Session` and `Application` level scope with caution. See pages 222 to 229 to see other issues associated with using the `Application` object and `Session` objects.

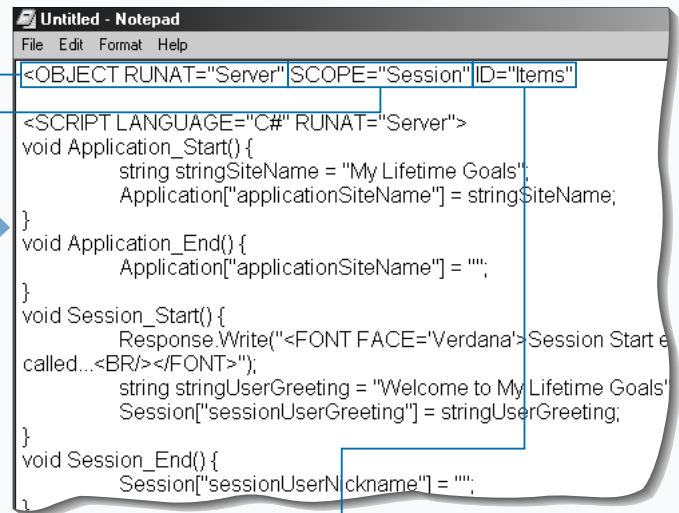
## USING SERVER-SIDE OBJECTS IN THE GLOBAL.ASAX FILE



- 1 Start the text editor to edit the `Global.asax` file.
- 2 Click `File` → `Open`.
- 3 Click  to select the `Code Templates` directory and open the `DeclareServerSideObjectGlobalTemplate.aspx` file.

- 4 Click `Open` to open the template.

*Note: You can open another `Global.asax` file that you want to use as a server-side object.*



- 5 Click where you want to declare a server-side object and type `<OBJECT RUNAT="Server">`.

*Note: Objects must be declared outside the `<SCRIPT>` tags.*

- 6 Type `SCOPE=` and the scope of the object.

*Note: The scope of an object can be `Session`, `Application`, or `AppInstance`.*

- 7 Type `ID=` and the name you want to reference the object by within your ASP pages.

## Apply It

After a server-side object has been declared in the `Global.asax` properly, you can access the object from any ASP page in your application. You reference the object by its ID. For example, if a `System.Collections.ArrayList` object has been declared in the `Global.asax` file with an ID of `Items`, you can use the object on any ASP page.

### TYPE THIS:

```
<OBJECT RUNAT="Server" ID="Items"
  CLASS="System.Collections.ArrayList" />
```

### RESULT:

After this object is placed in the `Global.asax` file, you can then work with the `Items` array list on any page.

```
Untitled - Notepad
File Edit Format Help

<OBJECT RUNAT="Server" SCOPE="Session" ID="Items"
CLASS="System.Collections.ArrayList"></OBJECT>

<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Application_Start() {
    string stringSiteName = "My Lifetime Goals";
    Application["applicationSiteName"] = stringSiteName;
}
void Application_End() {
    Application["applicationSiteName"] = "";
}
void Session_Start() {
    Response.Write("<FONT FACE='Verdana'>Session Start e
called...<BR/></FONT>");
    string stringUserGreeting = "Welcome to My Lifetime Goals";
    Session["sessionUserGreeting"] = stringUserGreeting;
}
void Session_End() {
```

- 8** Type `CLASS= "">`.
- 9** Between the quotation marks (""), type the kind of object you want to declare.

- 10** Click where you want to close the `<OBJECT>` tag and type `</OBJECT>`.

```
DeclareServerSideObjectGlobalTemplate.aspx - Notepad
File Edit Format Help
New Ctrl+N
Open... Ctrl+O
Save Ctrl+S

"Server" SCOPE="Session" ID="Items"
ollections.ArrayList"></OBJECT>

Save As...
Save As
Page Setup...
Print...
Save in: wwwroot
Exit
History
Desktop
My Documents
My Computer
My Network P...
File name: Global.aspx
Save as type: Text Documents (*.txt)
Encoding: ANSI
Save
Cancel
```

- 11** Click `File`  $\Rightarrow$  `Save As` to open the dialog box.

- 12** Click  $\square$  to select the folder where you want to store your file.

- 13** Type a name for the Web page.

- 14** Click `Save` to save the Web page.

*Note: Click the Yes button if you are prompted about replacing the file.*

*Note: A `System.Collections.ArrayList` variable with a name of `Items` will be available for each session created for the default Web site.*

# USING APPLICATION EVENT HANDLERS IN THE GLOBAL.ASAX FILE

You have eighteen supported application events in ASP.NET Applications for writing application-specific code. You also have the capability to create your own custom application level events. Typically, you will use the built-in events. You can use any of these built-in events by implementing the event handlers for these events in the `Global.asax` file. This is done in the `Global.asax` file with the following naming convention

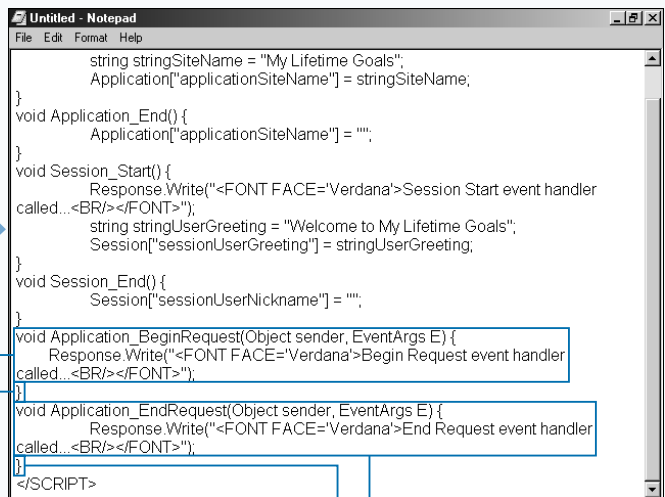
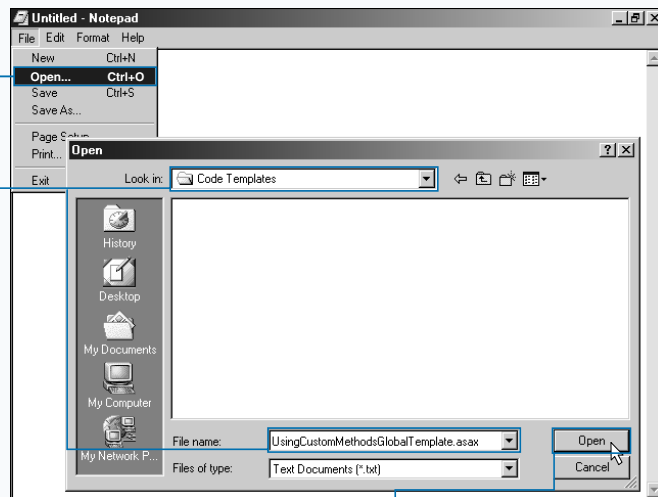
`Application_EventName (appropriateevent arguments signature)`.

Usually, you will add parameters to get more contextual information on the event. This is a typical event handler signature that includes parameters —

`Application_EventName (Object sender, EventArgs e)`. The sender tells you who raised this event and the `EventArgs` gives you further context about why the event was raised. The sender and `EventArgs` are not required parameters of the event handler, but it is good practice to include these parameters.

Some applications' events are raised for each request to the application, and others just occur under special conditions. For example, you can also use the `BeginRequest` and `EndRequest` events for code that you want to execute each time a user makes a request. Place this in the `Global.asax` file between the `<SCRIPT>` tags.

## USING THE BEGINREQUEST AND ENDREQUEST METHODS



**1** Start the text editor to edit the `Global.asax` file.

**2** Click `File` → `Open`.

**3** Navigate to the Code Templates directory and open the `DeclareServerSideObjectGlobalTemplate.aspx` file.

**4** Click `Open` to open the template.

*Note: You can open another `Global.aspx` file if you want.*

**5** To create the event handler for request, click between the `<SCRIPT>` and `</SCRIPT>` tags, type **void Application\_BeginRequest**(`{`, and press Enter.

**6** Type `}` to finish the event handler and press Enter.

**7** To create the event handler to end the request, click between the `<SCRIPT>` and `</SCRIPT>` tags, type **void Application\_EndRequest**(`{`, and press Enter.

**8** Press `Tab` to indent, type `}` to finish the event handler, and press Enter.

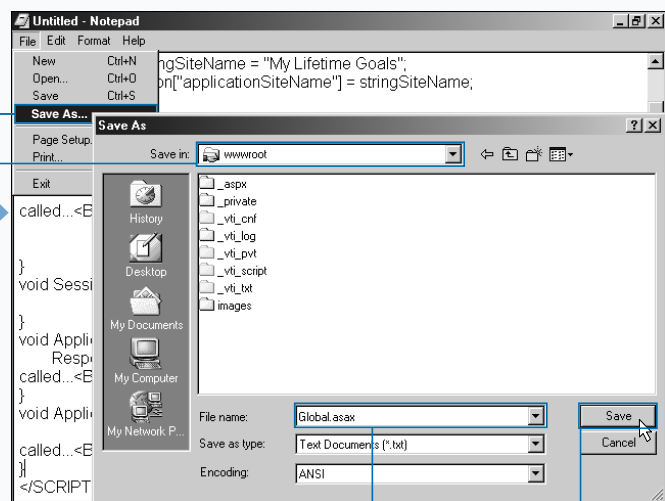
**Extra**

ASP.NET uses a pool of `HttpApplication` instances to service requests made by your application. These managed instances handle their requests for the lifetime of the request. With ASP.NET, you can write code that will run when `HttpApplication` instances are created and destroyed.

You can write code in the `Global.asax` file to override the creation and destruction methods for the `HttpApplication` instance, the `Init`, and the `Dispose` methods. The `Init` method executes for each `HttpApplication` instance that is created, whereas the `Dispose` method executes when the `HttpApplication` instance is destroyed.

**Example:**

```
<%
public void Init(){
    "Init override code goes here
}
public void Dispose()
    " Dispose override code goes here
}
%>
```



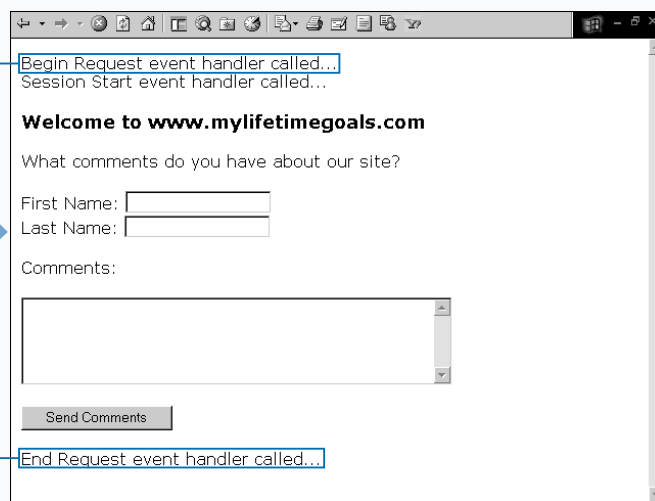
**9** Click **File** → **Save As** to open the dialog box.

**10** Click **wwwroot** to select the folder where you want to store your file.

**11** Type a name for the Web page.

**12** Click **Save** to save the Web page.

*Note: Click the Yes button if you are prompted about replacing the file.*



**13** Request an ASP.NET Web page from your site to initiate the creation of a new application.

The `BeginRequest` and `EndRequest` methods are called on the page.

*Note: In this example, the `Comments.aspx` file was requested.*

*Note: The Web browser is in full-screen mode in this screenshot so you can see all of the event messages.*

# USING APPLICATION STATE

You can share global information throughout your application by using the `HttpApplicationState` class. This class stores state in `Application` variables that are available to all users of the site. An instance of the `HttpApplicationState` class is created the first time any user requests a URL resource from within your ASP.NET application.

Creating and using `Application` variables is very simple. However, you need to weigh the benefits of simplicity against the cost of memory. The memory allocated for these application level variables is not released until the Web application is shut down or the variable is removed or replaced programmatically.

`Application` variables should only hold information that is necessary for all users of the site. You will also find it beneficial to place objects that take a long time to instantiate and are used frequently in `Application` variables. If the instantiation is expensive, you want to keep the object around after the client is done using the object, so the next client does not pay the same instantiation penalty. This being said, you still need to limit the number of objects and other variable types stored in the application, due to the constraint of available memory on your server.

## USING APPLICATION STATE

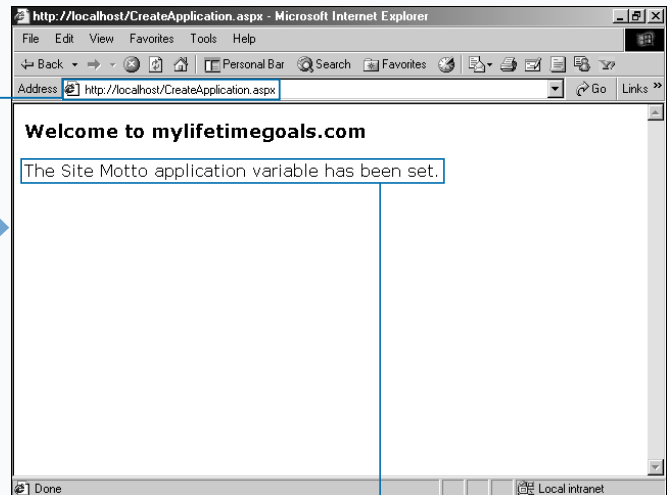
```

CreateApplicationTemplate.aspx - Notepad
File Edit Format Help
<%@Page Language="C#"%>
<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
<%
Application["applicationSiteMotto"] = "Our goal is to help you reach yours";
%>
The Site Motto application variable has been set

</FONT>
</BODY>
</HTML>

```



**1** Open `CreateApplicationTemplate.aspx` from the Code Templates directory, click where you want to create application information, and type `Application[""]=`.

**2** Between the quotation marks (" "), type the name of the `Application` variable you want to create.

**3** After the equal sign (=), type a value for the `Application` variable in quotes.

**4** Save the page as `CreateApplication.aspx` to the Default Web site and display the ASP.NET page in a Web browser.

The `Application` variable is set, and a message appears.

*Note: After the page is executed, you can access the information stored in the `Application` variable.*

**Extra**

Because everyone can access the `Application` object, you should lock the object when updating. When you finish updating the variable, you can unlock the variable to free it for further updating. It is not required to unlock (it will be automatically unlocked when the page goes out of scope), but it is good to explicitly unlock.

**Example:**

```
<%
    Application.Lock
    Application["applicationSiteName"] = "My Sample Site";
    Application.Unlock
%>
```

```
ReadApplicationTemplate.aspx - Notepad
File Edit Format Help
<%@Page Language="C#"%>
<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

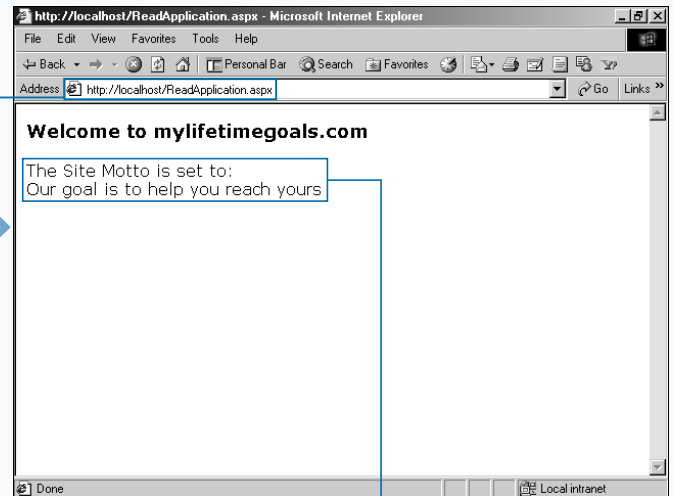
<H3>Welcome to mylifetimegoals.com</H3>
The Site Motto is set to:<BR/>
<%
Response.Write(Application["applicationSiteMotto"].ToString());
%>

</FONT>
</BODY>
</HTML>
```

**5** Open `ReadApplicationTemplate.aspx` from the Code Templates directory, click where you want to access application information, and type `Application[""]`.

**6** Between the quotation marks (" "), type the name of the `Application` variable you want to read.

**7** Use the `ToString()` function to convert the variable to a string and place a `Response.Write ()`; around code that accesses the `Application` variable.



**8** Save the page as `ReadApplication.aspx` and display the ASP.NET page in a Web browser.

The `Application` variable's value appears.

CONTINUED

# USING APPLICATION STATE

All users of your site share `Application` variables. You can create `Application` variables on any ASP.NET page on your site. However, programmers often use the `Application_Start` event handler to create your `Application` variables.

After you create the variables, you can read them from any page on the site. Because of the scope of `Application` variables, you should generally practice using `Application` variables for read-only data that is accessed often across the entire site. `Application` settings and common lookup values are good candidates for `Application` variables. For example, you can store state codes in an

`Application` variable. Note that you can also update and delete `Application` variables from any page on your Web site at any time.

Synchronization support is not built into `Application` variables. You must explicitly use the `Lock` and `Unlock` methods provided by the `HttpApplicationState` class in order to synchronize requests in a multi-user environment. Note that this can impact scalability by blocking other requests to the `Application` when you are updating or reading a value with a lock in place. Another scalability issue of using the `Application` object is that its state can not be shared across multiple servers.

## USING APPLICATION STATE (CONTINUED)

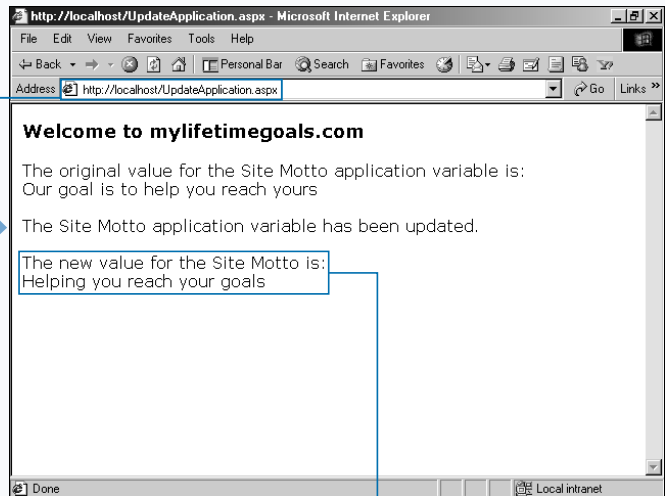
```
UpdateApplicationTemplate.aspx - Notepad
File Edit Format Help
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
<%
Application["applicationSiteMotto"] = "Our goal is to help you reach yours";
%>
The original value for the Site Motto application variable is:<BR/>
<%
Response.Write(Application["applicationSiteMotto"].ToString());
Application["applicationSiteMotto"] = "Helping you reach your goals";
%>
<BR/><BR/>
The Site Motto application variable has been updated.<BR/><BR/>
The new value for the Site Motto is:<BR/>
<%
Response.Write(Application["applicationSiteMotto"].ToString());
%>
</BODY>
</HTML>
```

**9** Open `UpdateApplicationTemplate.aspx` from the Code Templates directory.

**10** Click where you want to update the `Application` variable and type `Application["applicationSiteMotto"] = "";`

**11** Between the quotation marks (" "), type the updated value.



**12** Save the page as `UpdateApplication.aspx` to the default Web site and display the ASP.NET page in a Web browser.

The Web browser displays the result of updating the application information.



**Apply It**

You can treat the `Application` variables as a collection. This enables you to access all of the variables in your application in a looping structure.

**TYPE THIS:**

```
<%
    Application["applicationSiteName"] = "My Lifetime Goals";
    Application["applicationSiteMotto"] = "Our goal
    is to help you reach yours";
%>

The Application variables have been set.<BR/><BR/>
The Application variables are:<BR/><BR/>

<%
int intCount = Application.Count.ToInt32();
for ( int i= 0; i < intCount; i++ ) {
    Response.Write (Application.GetKey(i).ToString());
    Response.Write (" = ");
    Response.Write (Application.Get(i).ToString());
    Response.Write("<BR/>");
}
%>
```

**RESULT:**

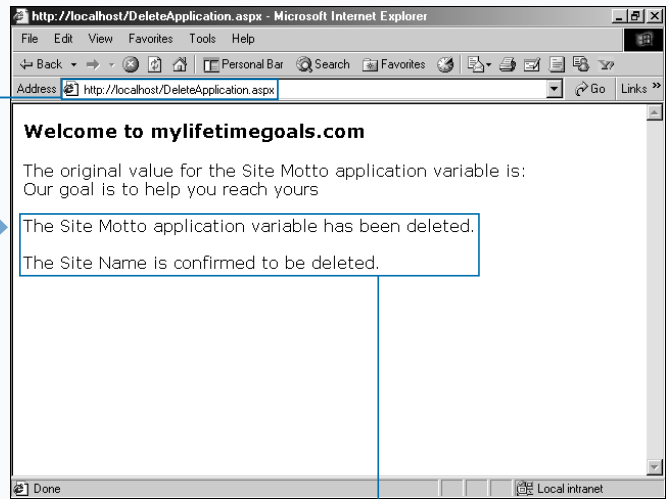
The application variables have been set.

The application variables are:

applicationSiteName = My Lifetime Goals

applicationSiteMotto = Our goal is to help you reach yours

```
File Edit Format Help
<H3>Welcome to mylifetimegoals.com</H3>
<%
Application["applicationSiteMotto"] = "Our goal is to help you reach yours";
%>
The original value for the Site Motto application variable is:<BR/>
<%
Response.Write(Application["applicationSiteMotto"].ToString());
Application.Contents.Remove("applicationSiteMotto");
%>
<BR/><BR/>
The Site Motto application variable has been deleted.<BR/><BR/>
<%
if (Application["applicationSiteMotto"] != null) {
    Response.Write("The Site Name was not confirmed to be deleted.");
}
else {
    Response.Write("The Site Name is confirmed to be deleted.");
}
%>
```



- 13 Open `DeleteApplicationTemplate.aspx` from the Code Templates directory.
- 14 Click where you want to delete the `Application` variable and type `Application.Contents.Remove("")`;

- 15 Between the quotation marks (" "), type the name of the variable you want to delete.

- 16 Save the page as `DeleteApplication.aspx` to the default Web site and display the ASP.NET page in a Web browser.

- The Web browser displays the result of deleting the application information.

# USING SESSION STATE

You can manage user-specific information from page to page in your application by using the `HttpSessionState` class. This class stores state in `Session` variables that are tied back to the requesting client. An instance of the `HttpApplicationState` class is created the first time the client requests a URL resource from within your ASP.NET application or when the client requests a URL resource after the session has expired or has been programmatically abandoned.

Session state management is critical to building Web applications that rely on information that crosses from one page to the next. Page personalization demonstrates the benefits of using `Session` variables. For personalization, you might load a user's

preferences at the beginning of his session and access the `Session` variables on each page request to customize the user's page presentation.

Starting the session will initiate the `Session_Start` event handler. The `Session_Start` event handler is the most common location for initializing `Session` variables (`Session["sessionvariable"]="some value";`). After a `Session` variable has been initialized in the `Global.asax` file or any page in the Web application, you can read this data from any page in the Web application (`string strStorage = Session["sessionvariable"];`). Using `Session` variables are not the only way to manage session data, but they are one of the easiest ways.

## USING SESSION STATE

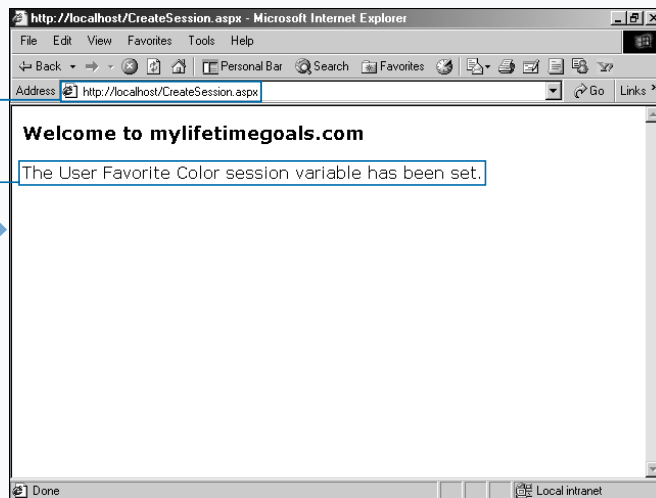
```

<%@Page Language="C#" Debug="true" %>
<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
<%
Session["sessionUserFavoriteColor"] = "Blue";
%>
The User Favorite Color session variable has been set.

</FONT>
</BODY>
</HTML>

```



**1** Open `CreateSessionTemplate.aspx` from the Code Templates directory and type `Session[""] = .`

**2** Between the quotation marks (" "), type the session variable you want to create.

**3** After the equal sign (=), type a value for the session variable in quotes.

**4** Save the page as `CreateSession.aspx` to the Default Web site and display the ASP.NET page in a Web browser.

*Note: After this page has been executed, you can access the information stored in the session variable.*

## Apply It

The `Session` object has many properties that you can configure. One of the important properties is the `Session.Timeout`. This value determines the amount of time that a user can go idle on your site without having to obtain a new session. You can set this property and read from this property.

### TYPE THIS:

```
<%
    Session.Timeout = 5;
    Response.Write("The Session Timeout is currently " + Session.Timeout +
" minutes.");
%>
```

### RESULT:

The Session Timeout is currently 5 minutes.

```
Untitled - Notepad
File Edit Format Help
<%@Page Language="C#"%>
<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
The User Favorite Color session variable is set to:
<%
Response.Write(Session["sessionUserFavoriteColor"].ToString());
%>
</FONT>
</BODY>
</HTML>
```

```
http://localhost/ReadSession.aspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Personal Bar Search Favorites
Address http://localhost/ReadSession.aspx Go Links
Welcome to mylifetimegoals.com
The User Favorite Color session variable is set to: Blue
Done Local intranet
```

**5** Open `ReadSessionTemplate.aspx` from the Code Templates directory and click where you want to access application information and type `Session[""]`.

**6** Between the quotation marks (" "), type the session variable you want to read.

**7** For the value to appear as output on the Web page, use the `ToString()` function to convert the variable to a string and place a `Response.Write()`; around code that accesses the `Application` variable.

**8** Save the page as `ReadSession.aspx` to the Default Web site and display the ASP.NET page in a Web browser.

The Web browser displays the result of reading the application information.

CONTINUED

# USING SESSION STATE

Web applications on IIS, by design, do not retain state as the user goes from page to page on the site. Therefore, you must have some mechanism of tracking information from page to page. Using the `Session` object is one of the easiest ways to manage user-specific information.

Session state management has improved with ASP.NET Applications. `Session` objects in ASP 3.0 could not be shared across Web forms, and therefore, many developers would avoid the use of `Session` in their applications.

To overcome the session state issues in ASP 3.0, ASP.NET gives you three options for session state

management. The first option is *in-process* (runs in the same memory space as ASP.NET). This gives the best performance if you are only going to deploy to one Web server. The second option is *SQL Server*. This is a persistent storage mechanism that has the slowest performance of all the options, but it is the most resilient to Web site failures (for example, power outage) and can be used in Web forms. The third option is *out-of-process*. This falls between *in-process* and *SQL Server* in terms of performance, and it can also be configured for Web forms. This option holds session data in volatile memory, so it is not as resilient as the *SQL Server* option.

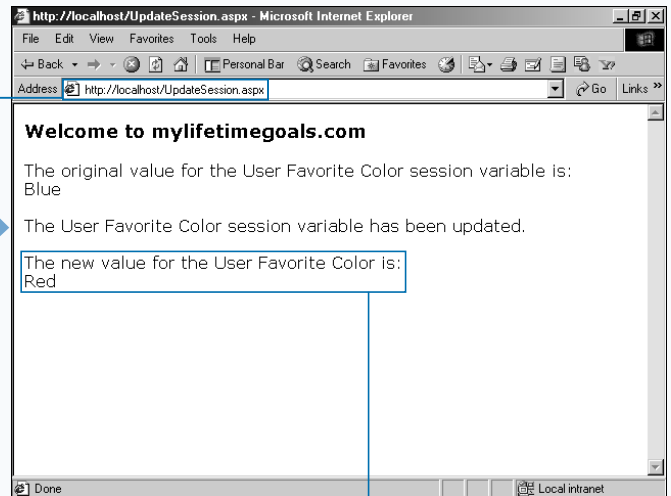
## USING SESSION STATE (CONTINUED)

```

Untitled - Notepad
File Edit Format Help
<%@Page Language="C#" %>
<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
The original value for the User Favorite Color session variable is:<BR/>
<%
Response.Write(Session["sessionUserFavoriteColor"].ToString());
Session["sessionUserFavoriteColor"] = "Red";
%>
<BR/><BR/>
The User Favorite Color session variable has been updated.<BR/><BR/>
The new value for the User Favorite Color is:<BR/>
<%
Response.Write(Session["sessionUserFavoriteColor"].ToString());
%>

```



**9** Open `UpdateSessionTemplate.aspx` from the Code Templates directory.

**10** Click where you want to update the session variable and type `Session["sessionUserFavoriteColor"] = "";`

**11** Between the quotation marks (" "), type the updated value.

**12** Save the page as `UpdateSession.aspx` to the Default Web site and display the ASP.NET page in a Web browser.

The Web browser displays the result of updating the session information.

## Apply It

You can treat the session variables as a collection. This enables you to access all of the variables in your application in a looping structure.

### TYPE THIS:

```
<%
    Session["sessionUserFavoriteColor"] = "Blue";
    Session["sessionUserSecondFavoriteColor"] =
        "Green";

    Session["sessionUserThirdFavoriteColor"] = "Red";
%>

The User Favorite Color Session variables have been
set.<BR/><BR/>

The User Favorite Color Session variables
are:<BR/><BR/>
<%
System.Collections.IEnumerator SessionEnumerator =
    Session.GetEnumerator();
while ( SessionEnumerator.MoveNext() ) {

Response.Write(SessionEnumerator.Current.ToString());
Response.Write(" = ");
Response.Write(Session[SessionEnumerator.Current.
    ToString()].ToString());
Response.Write("<BR/>");

}
%>
```

### RESULT:

The User Favorite Color Session variables have been set.

The User Favorite Color Session variables are:

sessionUserFavoriteColor = Blue

sessionUserSecondFavoriteColor = Green

sessionUserThirdFavoriteColor = Red

```
Untitled - Notepad
File Edit Format Help
<%@Page Language="C#"%>
<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
<%
Session.Contents.Remove("sessionUserFavoriteColor");
%>
The User Favorite Color session variable has been deleted.<BR/><BR/>
<%
if (Session["sessionUserFavoriteColor"] != null) {
    Response.Write("The User Favorite Color was not confirmed to be deleted.");
}
else {
    Response.Write("The User Favorite Color is confirmed to be deleted.");
}
%>
```

**13** Open `DeleteSessionTemplate.aspx` from the Code Templates directory.

**14** Click where you want to delete the Application variable and type `Session.Contents.Remove("");`.

**15** Between the quotation marks (" "), type the name of the variable you want to delete.

```
http://localhost/DeleteSession.aspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Home Personal Bar Search Favorites
Address http://localhost/DeleteSession.aspx Go Links
Welcome to mylifetimegoals.com
The User Favorite Color session variable has been deleted.
The User Favorite Color is confirmed to be deleted.
Done Local intranet
```

**16** Save the page as `DeleteSession.aspx` to the default Web site and display the ASP.NET page in a Web browser.

The Web browser displays the result of deleting the session information.

# WORK WITH COOKIES

You can use ASP.NET to create *cookies* from an ASP.NET page. When the user views the page, the cookie is stored as a small text file on the user's computer. A cookie consists of a key, which indicates the name of the cookie, and a value, which is the information stored in the cookie.

When you create a cookie, you should specify when the cookie will expire. By default, a cookie is usually deleted as soon as the user closes his or her Web browser. Setting an expiration date for a cookie enables the cookie to store information for longer periods of time. Most Web browsers store all the

cookies they receive in one folder. The storage location depends on the Web browser installed on the computer.

Typically, you will write the cookie to the user's browser before you begin sending any HTML in the response to the client. If you write a cookie midway through sending back a response, you may get an error. This depends on whether you have buffering of responses enabled on your page or for the application. Buffering the response for the entire application can be set in the IIS MMC. For IIS 5.0, buffering is turned on by default.

## WORK WITH COOKIES

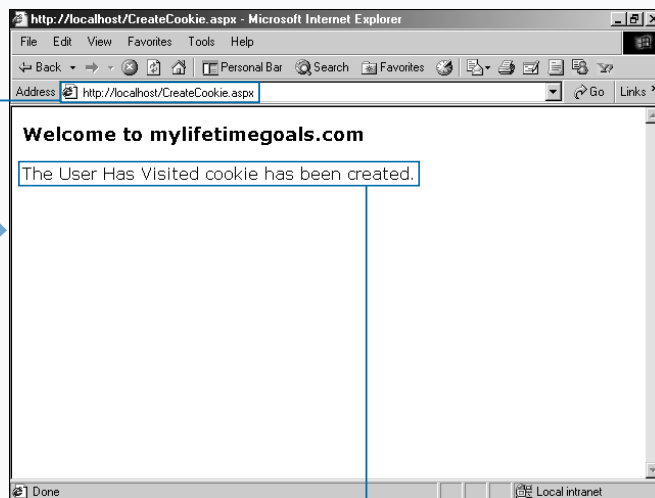
```

Untitled - Notepad
File Edit Format Help
<%@Page Language="C#"%>
<%
HttpCookie cookieUserInfo = new HttpCookie("cookieUserHasVisited");
cookieUserInfo.Value = "Yes";
Response.AppendCookie(cookieUserInfo);
%>
<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
The User Has Visited cookie has been created.

</FONT>
</BODY>
</HTML>

```



**1** Open `CreateCookieTemplate.aspx` from the Code Templates directory, click where you want to create a cookie, and type `HttpCookie cookieUserInfo = new HttpCookie("cookieUserHasVisited");` and press Enter.

**2** Set the cookie value by typing `cookieUserInfo.Value = "Yes";`.

**3** To write the cookie to the Web browser, type `Response.AppendCookie(cookieUserInfo);`.

**4** Save the page as `CreateCookie.aspx` to the Default Web site and display the ASP.NET page in a Web browser.

The Web browser displays a message about creating a cookie.

*Note: The cookie remains available until the Web browser is closed.*

## Apply It

If you do not set an expiration date for a cookie, the Web browser stores the cookie only until the user closes the Web browser. In most cases, you should keep this information for a longer period of time. To do this, you can use the `Expires` property of your cookie. You can use the `DateTime` object's `Now` property, along with some convenient functions that add time to the current date and time.

### TYPE THIS:

```
<%
HttpCookie cookieUserInfo = new
HttpCookie("cookieUserHasVisited");
cookieUserInfo.Value = "Yes";
cookieUserInfo.Expires =
DateTime.Now.AddDays(7).ToString();
Response.AppendCookie(cookieUserInfo);
%>
```

### RESULT:

The cookie is set for a week.

### TYPE THIS:

```
<%
HttpCookie cookieUserInfo = new
HttpCookie("cookieUserHasVisited");
cookieUserInfo.Value = "Yes";
cookieUserInfo.Expires =
DateTime.Now.AddMonths(1).ToString();
Response.AppendCookie(cookieUserInfo);
%>
```

### RESULT:

The cookie is set for a month.

```
Untitled - Notepad
File Edit Format Help
<%@Page Language="C#"%>
<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
The User Has Visited cookie is set to: <BR/>
<%
HttpCookie cookieUserInfo = Request.Cookies["cookieUserHasVisited"];
if (cookieUserInfo != null) {
    Response.Write(Request.Cookies["cookieUserHasVisited"].Value);
}
else
{
    Response.Write("No");
    Response.Write("<BR/>");
    Response.Write("Actually, it was not set so it was null");
}
%>
</FONT>
```

**5** Open `ReadCookie.Template.aspx` from the Code Templates directory.

**6** Read and display the cookie contents by typing `Response.Write(Request.Cookies["cookieUserHasVisited"].Value);`.

http://localhost/ReadCookie.aspx - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://localhost/ReadCookie.aspx

Welcome to mylifetimegoals.com

The User Has Visited cookie is set to:  
Yes

Done Local intranet

**7** Save the page as `ReadCookie.aspx` to the default Web site and display the ASP.NET page in a Web browser.

The Web browser displays a message about the contents of the cookie.

**8** Open `CreatCookie.CollectionTemplate.aspx` from the Code Templates directory.

CONTINUED

# WORK WITH COOKIES

Cookies are an alternative to using `Session` objects if you want to share data from page to page. If this data is very sensitive and you do not want your user to see this data, cookies are not a good option. Cookies are stored in plain text on the user's local machine.

After you have created the cookie on the user's machine, you can then access the cookie from another page. After the ASP.NET page finds the value of the cookie, you can have the page perform an action depending on the value. A `Request.Cookies` statement enables you to read a cookie. You must

know the name of the cookie you want to read. If the cookie you want to read does not exist on the user's computer, the value of the cookie will be null.

A cookie can use subkeys to store several related values. For example, you could create a cookie that stores the user's font preferences. For this cookie, you could store both the preference for the font size and for the font name. With subkeys, you can do this using one cookie. To add a subkey to the cookie, use the `Values.Add` method for the `HttpCookie` object.

## WORK WITH COOKIES (CONTINUED)

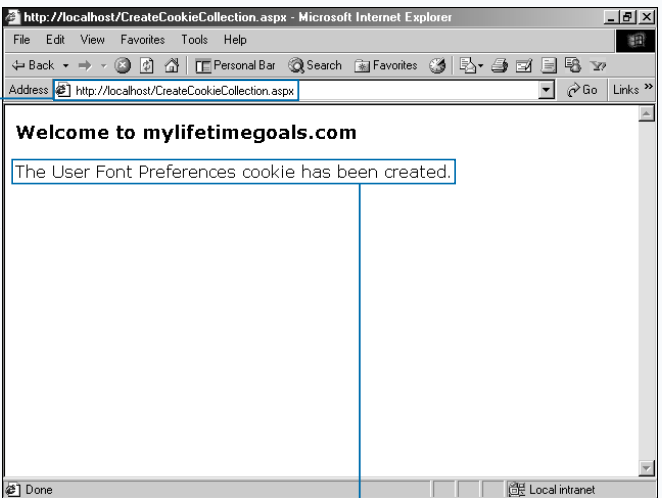
```

Untitled - Notepad
File Edit Format Help
<%@Page Language="C#"%>
<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
<%
HttpCookie cookieUserInfo = new HttpCookie("cookieUserFontPreferences");
cookieUserInfo.Values.Add("FontSize","8pt");
cookieUserInfo.Values.Add("FontName","Verdana");
Response.AppendCookie(cookieUserInfo);
%>
The User Font Preferences cookie has been created.

</FONT>
</BODY>
</HTML>

```



**9** Click where you want to create a cookie and type `HttpCookie cookieUserInfo = new HttpCookie("cookieUserFontPreferences");`.

**10** Set the cookie subkey by typing `cookieUserInfo.Values.Add("FontSize","8pt");`.

**11** Set another the cookie subkey by typing `cookieUserInfo.Values.Add("FontName","Verdana");`.

**12** To write the cookie to the Web browser, type `Response.AppendCookie(cookieUserInfo);`.

**13** Save the page as `CreateCookie.aspx` to the Default Web site and display the ASP.NET page in a Web browser.

The Web browser displays a message about creating a cookie.

*Note: The cookie remains available until the Web browser is closed.*



## Apply It

You can loop through the subkeys of a cookie, which enables you to work with the subkeys as a collection, perhaps displaying the names and values of the subkeys in the cookie.

### TYPE THIS:

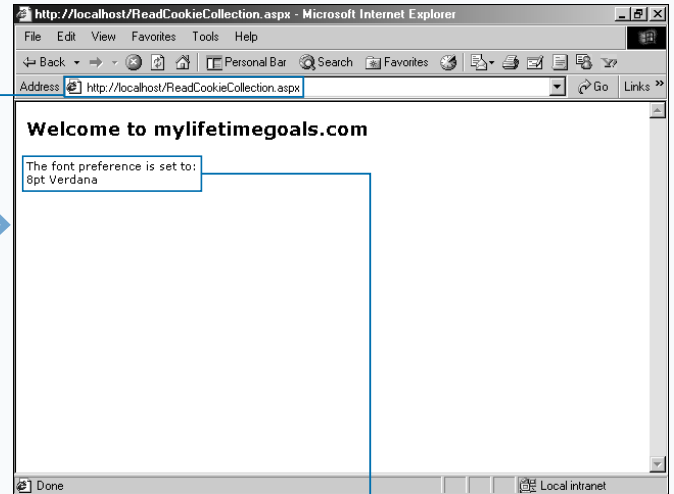
```
<%
HttpCookieCollection cookie collectionAll
For (int loop=0; loop < cookie collectionAll.Count; loop ++
    {HttpCookie cookie = cookie collectionAll(loop);
      If (cookie.HasKeys)
        {NameValuePairCollection nameValueCollection = new
NameValuePairCollection(cookie.Values);
          String.[] StringValueNames = nameValueCollection.All Keys;
          String {} StringValues = nameValueCollection.All;
        }
      }
%>
```

### RESULT:

```
FontSize = 8pt
FontName = Verdana
```



```
Untitled - Notepad
File Edit Format Help
<%@Page Language="C#" %>
<%
HttpCookie cookieUserInfo = Request.Cookies["cookieUserFontPreferences"];
string stringFontSize = "";
string stringFontName = "";
if (cookieUserInfo != null) {
    stringFontSize = cookieUserInfo.Values["FontSize"];
    stringFontName = cookieUserInfo.Values["FontName"];
}
else {
    stringFontSize = "10pt";
    stringFontName = "Arial";
}
%>
<HTML>
<HEAD>
</HEAD>
<STYLE>
body {
font: <%=stringFontSize%> <%=stringFontName%>;
}
</STYLE>
```



**14** Open `ReadCookieCollectionTemplate.aspx` from the Code Templates directory, click where you want to read a cookie, and type `HttpCookie cookieUserInfo = Request.Cookies["cookieUserFontPreferences"];`.

**15** Read the contents of the subkey in the cookie into a variable that will style the page by typing `stringFontSize = cookieUserInfo.Values["FontSize"];`.

**16** To read the second subkey, type `stringFontName = cookieUserInfo.Values["FontName"];`.

**17** Save the page as `ReadCookieCollection.aspx` to the Default Web site and display the ASP.NET page in a Web browser.

The Web browser displays a message about the contents of the cookie.

# WORK WITH PAGE STATE

You can use *Page State* to store information that does not span multiple pages. This is appropriate for persisting data on a single page that is used across multiple requests of the same page.

To place a variable into Page State, use the syntax `ViewState['Name'] = Value`. To read the variable out of Page State, use the syntax `Variable = ViewState['Name']`. `ViewState` is inherited by the `Control` class. `Page` class and all `Web` controls are derived from the `Control` class and, therefore, have the ability to retain state for multiple requests of the same page.

The state bag is a data structure maintained by the page for retaining values between round trips to the server. By storing a value in the page's state bag, you automatically preserve it between round trips.

This section illustrates how to use page state to store the current step. Note the use of the `(IsPostBack!)` in the `Page_Load` event handler in the code. This code will only execute the first time the page is loaded because each subsequent request of the page is doing a postback to the page. The panel's `visible` property hides and displays the two panels on the page. For more information about panels, see page 112.

## WORK WITH PAGE STATE

```
Generic Template.aspx - Notepad
File Edit Format Help
<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
<FORM RUNAT="Server">
<ASP: PANEL ID="panelStep1" RUNAT="SERVER">
You are on Step <% Response.Write(ViewState["viewstateStep"]);%>. Are you ready to
set your goals? Click on the Continue button to go to Step 2.
</P>
<ASP:BUTTON ID="buttonContinue" RUNAT="Server" onClick="Button_OnClick"
TEXT="Continue">
</ASP:BUTTON>
</ASP: PANEL>
</FORM>
</FONT>
</BODY>
</HTML>
```

**1** Open the `Template.aspx` template from the directory.

**2** Type a heading to the page.

**3** Type a form control to the page.

**4** Type a panel to the page with an ID of `panelStep1`.

```
Generic Template.aspx - Notepad
File Edit Format Help
<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
<FORM RUNAT="Server">
<ASP: PANEL ID="panelStep1" RUNAT="SERVER">
You are on Step <% Response.Write(ViewState["viewstateStep"]);%>. Are you ready to
set your goals? Click on the Continue button to go to Step 2.
</P>
<ASP:BUTTON ID="buttonContinue" RUNAT="Server" onClick="Button_OnClick"
TEXT="Continue">
</ASP:BUTTON>
</ASP: PANEL>
<ASP: PANEL ID="panelStep2" RUNAT="SERVER" VISIBLE="False">
You are on Step <% Response.Write(ViewState["viewstateStep"]);%>.
</P>
</ASP: PANEL>
</FORM>
</FONT>
</BODY>
```

**5** Type a message about which step the user is on and output the value from the view state.

**6** Type a button control to the page that calls the `Button_OnClick` when clicked.

**7** Add another panel to the page with an ID of `panelStep2` and set the `visible` property to `False`.

**8** Add a message about which step the user is on and output the value from the view state.

## Extra

Page State information does not work across pages, this section features panels to display the page state information. You can test what happens when you use page state on two pages.

## TYPE THIS:

```
<%@Page Language="C#" %>
<HTML><HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
    void Page_Load(Object Src, EventArgs E) {
        if (!IsPostBack) ViewState["viewstateStep"] = 1;
    }
    void Button_OnClick(object Source, EventArgs e) {
        Response.Redirect("PageState_ai2.aspx");
    }
</SCRIPT></HEAD>
<BODY><FONT FACE="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
<FORM RUNAT="Server">
<ASP: PANEL ID="panelStep1" RUNAT="SERVER">
You are on Step <%
Response.Write(ViewState["viewstateStep"]);%>.
Are you ready to set your goals?
Click the Continue button to go to Step 2.
<P/>
<ASP: BUTTON ID="buttonContinue" RUNAT="Server"
onClick="Button_OnClick" TEXT="Continue"/>
<P/>
</ASP: PANEL>
</FORM></FONT></BODY>
</HTML>
```

## RESULT:

```
<%@Page Language="C#" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="
Server">
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">
<H3>Welcome to
mylifetimegoals.com</H3>
<FORM RUNAT='Server">
You are on Step <%
Response.Write(ViewState[
"viewstateStep"]);%>.
</FORM>
</FONT>
</BODY>
</HTML>
```

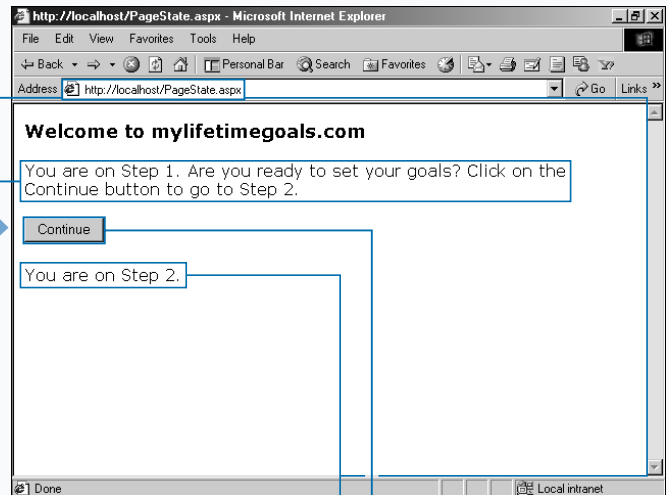
```
GenericTemplate.aspx - Notepad
File Edit Format Help
<%@Page Language="C#" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Page_Load(Object Src, EventArgs E) {
    if (!IsPostBack) ViewState["viewstateStep"] = 1;
}
void Button_OnClick(object Source, EventArgs e) {
    panelStep1.Visible = false;
    panelStep2.Visible = true;
    ViewState["viewstateStep"] = (int)ViewState["viewstateStep"] + 1;
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
<FORM RUNAT="Server">
<ASP: PANEL ID="panelStep1" RUNAT="SERVER">
You are on Step <% Response.Write(ViewState["viewstateStep"]);%>. Are you ready to
set your goals? Click on the Continue button to go to Step 2.
<P/>
```

9 Set the language for the page to C# using the Page directive.

10 Add the <Script> tag to set up some server-side code.

11 Add the Page\_Load event handler and initialize the view state variable.

12 Add the Button\_OnClick event handler and set the first panel's visible property to false and the second panel's visible property to true. Also, increment the view state variable by 1.



13 Save the file as PageState.aspx to the default Web site and request it from the Web server.

The Web browser displays a message about being on step 1.

14 Click the Continue button.

The Web browser displays a message about being on step 2.

Note that you stay on the same page.

# ADD APPLICATION SETTINGS

The `web.config` file gives you a maintainable, convenient, and secure means to store configuration information. The `web.config` file is an XML file that stores information used to customize application level settings across your entire ASP.NET site. For example, you could store a connection string to a database.

The `web.config` file needs to be located in the parent directory of the ASP.NET Web site for the pages that use the application configuration information. You do not have to place the file in the root directory of your ASP.NET application. You can place a `web.config` file in any directory of the Web

site and when a Web page accesses configuration information it will look in the parent directory. If the `web.config` file is not found in the parent directory, it will work its way up the directory structure until it hits the root of the Web site.

To put your own custom application settings in the `web.config` file you need to add the `<appSettings>` tag to the root `<configuration>` tag. When you have done this, you can add settings by using the `<add>` tags and specifying the name and value of the setting (for example, `<add key="pubs" value="server=(local);uid=sa;pwd=;database=pubs" />`).

## ADD APPLICATION SETTINGS

```
<configuration>
<appSettings>
<add key='pubs' value='server=localhost;uid=sa;pwd=;database=pubs' />
</appSettings>
</configuration>
```

**1** Open a new document in your text editor.

**2** Add a pair of `<configuration>` tags.

**3** Add a pair of `<appSettings>` tags.

**4** Add an `<add>` tag and set the `key` attribute `pubs` to the SQL connection string for connecting to the `pubs` database.

```
<%@ Import Namespace='System.Data' %>
<%@ Import Namespace='System.Data.SqlClient' %>
<%@ Import Namespace='System.Configuration' %>

<HTML>
<HEAD>
<SCRIPT LANGUAGE='C#' RUNAT='Server'>
protected void Page_Load(Object sender, EventArgs e) {
    SqlConnection sqlconnectionPubs = new
    SqlConnection("server=(local)\NetSDK;uid=QUser;pwd=QPassword;database=pubs");

    SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter("select title,
notes, price from titles where type='business'", sqlconnectionPubs);

    DataSet datasetTitles = new DataSet();
    sqldataadapterTitles.Fill(datasetTitles, "titles");

    datagridTitles.DataSource=datasetTitles.Tables["titles"].DefaultView;
    datagridTitles.DataBind();
}
</SCRIPT>
</HEAD>
</HTML>
```

**5** Save the file as `web.config`.

**6** Open the `DataGridTemplate.aspx` template from the Code Templates directory.

**7** Import the `System.Configuration` namespace.

**Extra**

Making changes to the `web.config` file causes it to be reloaded the next time a resource is requested. This will slow down your next request, but subsequent requests are not affected because the file is cached.

`web.config` files can contain very sensitive information that you do not want a user of your site to see. These settings could be connection strings, authorization information, MSMQ (Microsoft Message Queue) settings, and so on. By default, ASP.NET does not allow this file to be requested from the Web server. If users attempt to request a `web.config` file, they receive an HTTP access error, "This type of page is not served."

You can access configuration information by importing the `System.Configuration` namespace and running the following example code.

**Example:**

```
string stringPubsConnectionString =
ConfigurationSettings.AppSettings["pubs"];
```

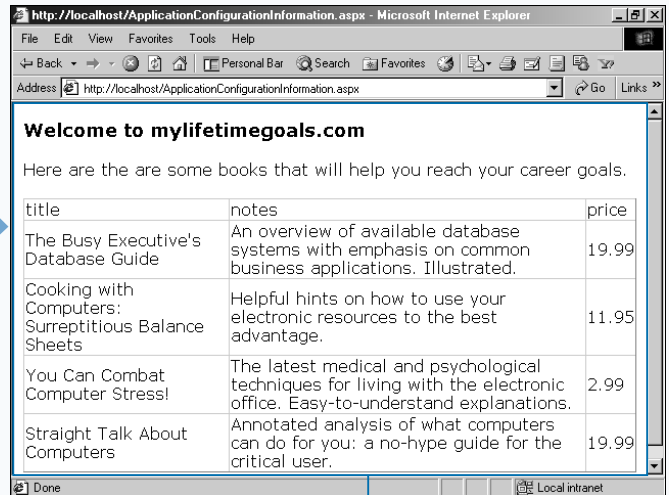
```
File Edit Format Help
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>

<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
protected void Page_Load(Object sender, EventArgs e) {
    string stringPubsConnectionString = ConfigurationSettings.AppSettings["pubs"];
    SqlConnection sqlconnectionPubs = new
    SqlConnection(stringPubsConnectionString);

    SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter("select title,
    notes, price from titles where type='business'", sqlconnectionPubs);

    DataSet datasetTitles = new DataSet();
    sqldataadapterTitles.Fill(datasetTitles, "titles");

    datagridTitles.DataSource=datasetTitles.Tables["titles"].DefaultView;
    datagridTitles.DataBind();
}
</SCRIPT>
```



**8** Create a `string` variable and initialize the variable using the `Configuration.AppSettings["pubs"]` property.

**9** Modify the creation of the SQL Connection to use the connection string read from the configuration file.

**10** Save the file and request it from the Web server.

The page appears using the SQL connection setting from the `web.config` file.

# SET STANDARD CONFIGURATION

You can use the standard configuration settings to specify how to configure your ASP.NET Web application. The `web.config` file is a central storage location for information that can apply to the entire site or just a section of the site. For example, you can configure how the `Session` state can be persisted on your Web site using the `<sessionState>` tag under the `<system.web>` section.

The `MODE` attribute can be set to determine where a user's session is stored (`mode="Inproc"`). There are three modes that you can choose from: `Inproc`, `StateServer`, and `SQLServer`. `Inproc` is the fastest and least durable mode that holds Session

state in the memory of the Web server process. Conversely, `SQLServer` is the slowest and most durable storage due to storage on a central SQL database store.

In the `web.config` file, you can place as many configuration settings as you desire. You are not required to stub in all the configuration settings. You are also not required to have a `web.config` file in your ASP.NET site. If your site accesses application settings and the `web.config` file does not exist, you will get a Web server error.

## SET STANDARD CONFIGURATION

```

<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Standard Configuration Information</H3>
Your user agent is:<P/>
<% Response.Write(Request.ServerVariables["HTTP_USER_AGENT"].ToString());
%> <P/>

</FONT>
</BODY>
</HTML>

```

- 1 Open the `GenericTemplate.aspx` template from the Code Templates directory.
- 2 Add a heading for the page.

- 3 Add a message to the user.
- 4 Write the `HTTP_USER_AGENT` server variable to the Web browser.

**Server Error in '/' Application**

**Compilation Error**

**Description:** An error occurred during the compilation of a resource required to service this request. Please review the specific error details below and modify your source code appropriately.

**Compiler Error Message:** BC30203: Expected identifier.

**Source Error:**

```

Line 8: <H3>Standard Configuration Information</H3>
Line 9: Your user agent is:<P/>
Line 10: <% Response.Write(Request.ServerVariables["HTTP_USER_AGENT"].ToString());
Line 11:

```

- 5 Save the file and request it from the Web server.

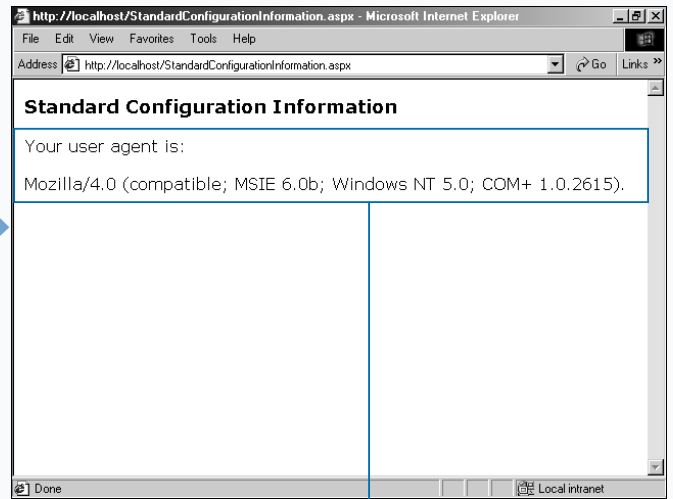
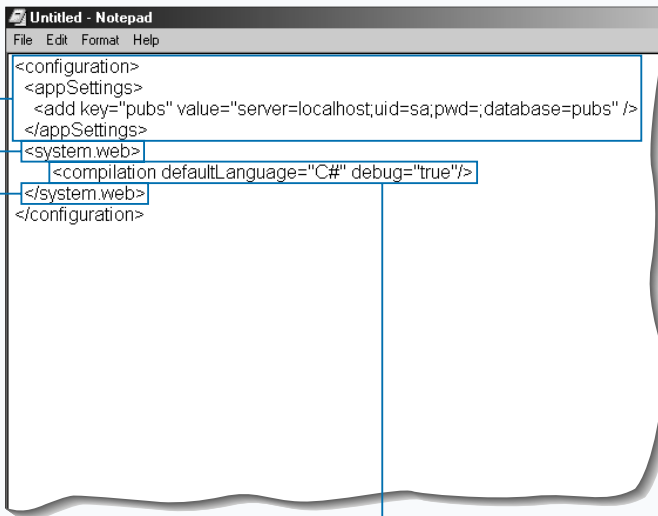
The error page appears because the code on the page was written in C# and C# is not set as the default language.

**Extra**

In the `<configuration>` tag, you can use `debug="true"` to enable ASP.NET debugging. You can specify the amount of time in seconds that ASP.NET should execute running an ASP.NET Web page before it times out and sends an error message about the timeout with the `<executionTimeout>` section.

**Example:**

```
<configuration>
  <system.web>
    <compilation defaultLanguage="C#"
      debug="true" />
    <httpRuntime executionTimeout="30" />
  </system.web>
</configuration>
```



**6** Create a `web.config` file.

**7** Add the `<system.web>` tags.

**8** Add the `<compilation>` tag and set the `defaultLanguage` attribute to C#.

**9** Save the file.

**10** Request the file from the Web server.

The page appears without problems because the default language is set to C#.

*Note: If you are creating a new web.config file, first add the <configuration> tags.*

# ADD CUSTOM SETTINGS

You can customize the `web.config` file with new sections. This makes the `web.config` file flexible enough to contain all of your configuration information in a structured way. The custom configuration section information is placed into two main areas in the configuration file.

The first area is for declaring names, groups, and handles for the custom sections. Place configuration section declarations in the `<configSections>` container tags. You will have `select` groups, `<sectionGroup>`, and `section` `<section>` tags for defining the structure of your custom configuration information. The `<sectionGroup>` is used to give hierarchy to your `<section>` tags. The `<section>`

tags in your declaration have two properties, `type` and `name`. The `type` attribute is the name of the class that reads the information, for example, `System.Configuration.NameValueSectionHandler` class is a structure for name value pairs. The `name` attribute is the name of the tag that contains the information the section handler will read.

Now that the sections and section groups are defined, you can store the custom configuration information by placing the `<sectionGroup>` and `<section>` tags under the `<configuration>` root node. With these tags, you will add the custom information in the `<section>` tag attributes.

## ADD CUSTOM SETTINGS

```

<configuration>
<configSections>
<sectionGroup name="system.web">
<section name="goalssetup"
type="System.Configuration.NameValueSectionHandler,System"/>
</sectionGroup>
</configSections>
<appSettings>
<add key="pubs" value="server=localhost;uid=sa;pwd=;database=pubs" />
</appSettings>
<system.web>
<compilation defaultLanguage="C#" debug="true"/>
<httpRuntime executionTimeout="30"/>
</system.web>
</configuration>

```

```

<configuration>
<configSections>
<sectionGroup name="system.web">
<section name="goalssetup"
type="System.Configuration.NameValueSectionHandler,System"/>
</sectionGroup>
</configSections>
<appSettings>
<add key="pubs" value="server=localhost;uid=sa;pwd=;database=pubs" />
</appSettings>
<system.web>
<goalssetup>
<add key="maxnumber" value="10"/>
</goalssetup>
<compilation defaultLanguage="C#" debug="true"/>
<httpRuntime executionTimeout="30"/>
</system.web>
</configuration>

```

**1** Create a `web.config` file.

**2** Add the `<configSections>` tags.

**3** Add the `<sectionGroup>` tag and set the name attribute to `system.web`.

**4** Add the `<section>` tag and set the `type` attribute to `System.Configuration.NameValueSectionHandler, System`.

**5** Add the `<goalssetup>` tag within the `<system.web>` tags.

**6** Add the `<add>` tag and set the key attribute to `maxnumber` and the value attribute to `10`.

*Note: If you are creating a new `web.config` file, first add the `<configuration>` tags.*



**Extra**

You may have noticed a couple of peculiar things about the source code for retrieving the value in the `web.config` file. Notice the use of `(String)` and `(NameValueCollection)`. These are classes defined for reading the configuration information. For example, the value retrieved from `Context.GetConfig("system.web/goalssetup")` is cast into a `NameValueCollection` type variable and then the value for the "maxnumber" is cast into a string.

You can access custom configuration information you use the `HttpContext` class in the `System.Web` namespace (`Context.GetConfig("system.web/goalssetup")`). The `GetConfig` gives you access to the `web.config` file. You need to specify the path to the node that you wish to work with, in this case, "system.web/goalssetup".

You can put this configuration information into a `Session` variable. This would make sense if you were modifying the value of the configuration information. For example, the `maxnumber` could be read into a `Session` variable when the user logs into the Web site. Based on the actions of the user, you could update the value of the `Session` variable.

```

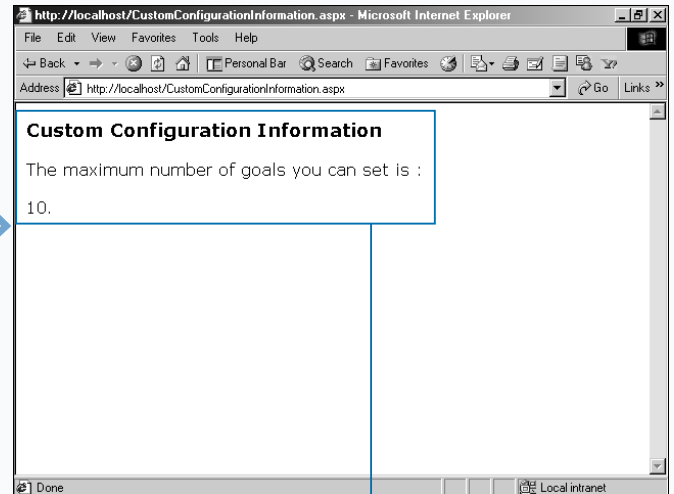
Untitled - Notepad
File Edit Format Help
<% @Page Language="C#" %>
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Custom Configuration Information</H3>
The maximum number of goals you can set is :<P/>
<% string stringMaxNumber = (String) ((NameValueCollection)
Context.GetConfig("system.web/goalssetup"))["maxnumber"];
Response.Write(stringMaxNumber); %>.<P/>

</FONT>
</BODY>
</HTML>

```



**7** Open the `GenericTemplate.aspx` template from the Code Templates directory.

**8** Add a heading for the page.

**9** Add a message to the user.

**10** Write the value of the `maxnumber` configuration.

**11** Save the file and request it from the Web server.

A message appears that displays the `maxnumber` custom setting from the `web.config` file.

# ENABLE PAGE-LEVEL DEBUGGING

If you are having problems with a page during development, you can turn page-level debugging on for the page to get more detailed error information, including the line number and the source code associated with the error. ASP.NET can configure debugging at the page level.

On the page you want to debug, add a `<@ Page debug="true" >` directive to the top of the page. The next time the page is requested and an error occurs on the page, detailed error information is displayed. Without this directive, you will get an error page that does not have any relative error information that can assist you in understanding why the error occurred.

Only pages that have the `debug` attribute set on the `@Page` directive are compiled into debug mode. If you want all pages in the site to run in debug mode, you need to update the `web.config` file. To enable page debugging application-wide, you need to add the compilation element under the `<system.web>` tag and set the `debug` attribute equal to `true`.

Make sure that you only turn on page debugging when necessary. Running applications in debug mode does incur a memory/performance overhead. For most cases you should not enable this in production.

## ENABLE PAGE-LEVEL DEBUGGING

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
protected void Page_Load(Object sender, EventArgs e) {
    SqlConnection sqlconnectionPubs = new
    SqlConnection("server=(local)\NetSDK;uid=QSUser;pwd=QSPassw
    s");

    SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter
    note price from titles where type='business", sqlconnectionPubs);

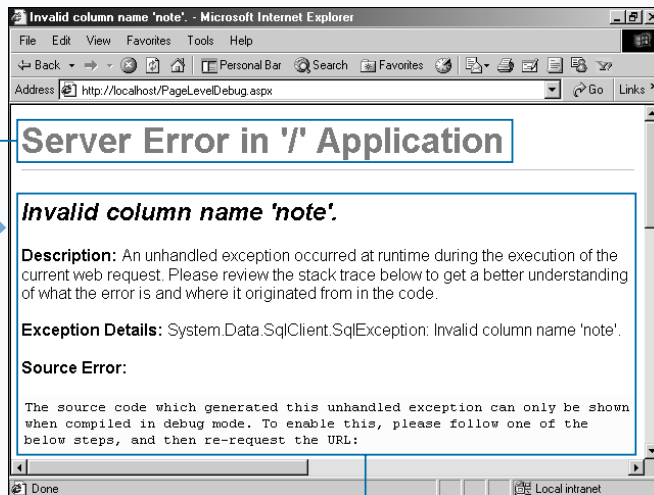
    DataSet datasetTitles = new DataSet();
    sqldataadapterTitles.Fill(datasetTitles, "titles");

    datagridTitles.DataSource=datasetTitles.Tables["titles"].D
    datagridTitles.DataBind();
  
```

**1** Open the `DatagridTemplate.aspx` template from the Code Templates directory.

**2** Create an error on the page by changing the SQL statement to an invalid SQL statement.

**3** Save the page and request it from the Web server.



The page displays and an error message appears.

A message appears from the server with a description of the error; however, no details are given as to on which line the error occurred.

**Extra**

When page-level debugging is set, you may receive several pieces of information to help you play detective when an error occurs. Here are the elements of the error message:

ELEMENT	DESCRIPTION
Description	A brief description of the error.
Exception Details	Indicates which exception was raised and describes exception.
Source Error	Displays a couple of lines of the source code before and after the line that generated the error.
Source File	Specifies the path and filename of the source file that generated the error.
Line	Indicates the line number of the error.
Stack Trace	Displays the call stack for the error.
Version Information	Displays the Runtime Build and the ASP.NET Build.

```

Untitled - Notepad
File Edit Format Help
<%@ Page Debug="true" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
protected void Page_Load(Object sender, EventArgs e) {
    SqlConnection sqlconnectionPubs = new
    SqlConnection("server=(local)\NetSDK;uid=QUser;pwd=QSPassw
    s");

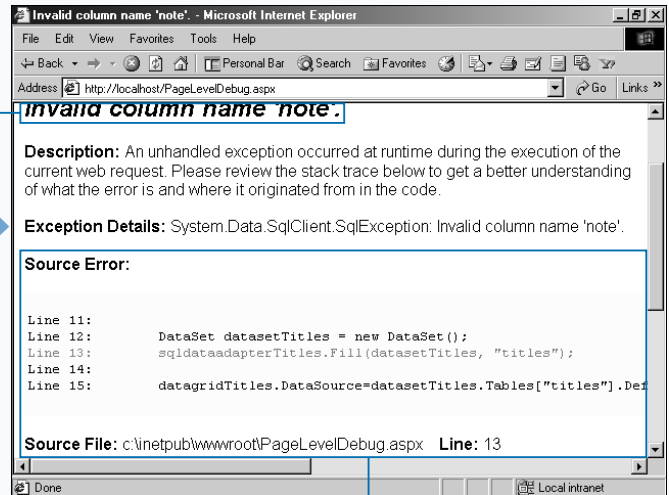
    SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter
    note, price from titles where type='business'", sqlconnectionPubs);

    DataSet datasetTitles = new DataSet();
    sqldataadapterTitles.Fill(datasetTitles, "titles");

    datagridTitles.DataSource=datasetTitles.Tables["titles"].D
    
```

**4** Return to the page you were creating and add the `@Page Debug` trace directive.

**5** Save the page and request it from the Web server.



The page displays and an error message appears.

A message appears from the server with a description of the error and the line number and the source code where the error occurred.

# ENABLE CUSTOM ERROR HANDLING

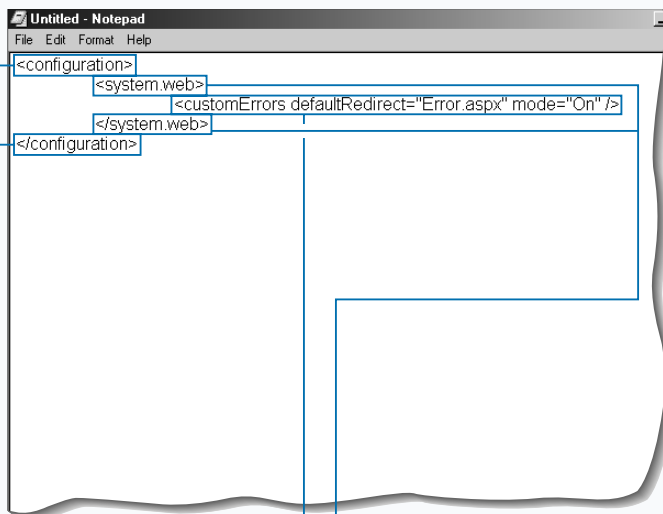
You can use the `web.config` file to specify a common error handling page(s) for your ASP.NET Web application. The error handler in the `web.config` file is declared in the `<customErrors>` tag under the `<system.web>` node.

In the `<customErrors>` tag, you can specify the Web page to direct users to when an error occurs with the `defaultRedirect` attribute. If you want to redirect users on a specified error code, you can use the `<error>` child node to redirect users to a page depending on the status code. For example, `<error statusCode="404" redirect="PageMissing.aspx" />` nested under the `<customErrors>` tag will redirect requests for

missing pages to an error-handling page that is only for missing pages. If this is the only `<error>` child node, then all errors except for 404 will be redirected to the page defined in the `<customErrors>` tag.

Another useful attribute of the `customErrors` element is the `mode`. There are three modes that can be set (On, Off, and RemoteOnly). The first two settings are self explanatory. The latter of these is for turning on custom error pages for remote users only. Locally logged-on users see the standard page debugging details like Source Error, Line, and Stack Trace. This enables an administrator to troubleshoot a problem without turning off the custom errors for outside users.

## ENABLE CUSTOM ERROR HANDLING



```

<configuration>
  <system.web>
    <customErrors defaultRedirect="Error.aspx" mode="On" />
  </system.web>
</configuration>
  
```

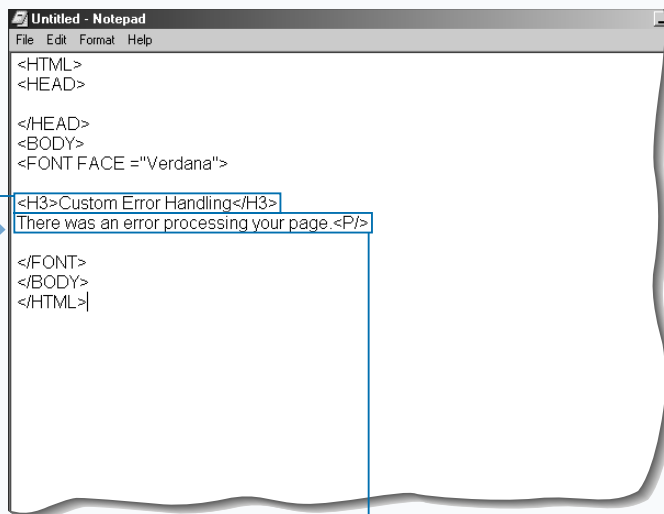
**1** Open a new document in your text editor.

**2** Start a `web.config` file by adding `<configuration>` tags.

**3** Add `<system.web>` tags.

**4** Add a `<customErrors>` tag and set the `defaultRedirect` attribute and the `mode` attribute.

**5** Save the file as `web.config` to the Web site.



```

<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3>Custom Error Handling</H3>
There was an error processing your page.<P/>
</FONT>
</BODY>
</HTML>
  
```

**6** Open the `GenericTemplate.aspx` template from the Code Templates directory.

**7** Add a heading to the page.

**8** Add an error message for the page.

**9** Save the page to the Web site.

**Apply It**

You can use the `QueryString` in the `Response` object to create an error page that gives more detail to the user.

**TYPE THIS:**

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

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Custom Error Handling</H3>
You had an error on <% Response.Write(Request.QueryString
["aspxerrorpath"].ToString()); %>.<P/>

</FONT>
</BODY>
</HTML>
```

**RESULT:**

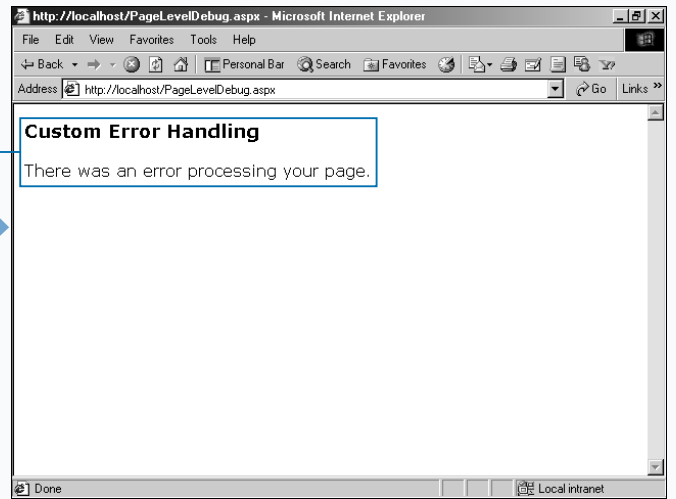
Custom Error Handling  
You had an error on /PageWithError.aspx.

```
Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
protected void Page_Load(Object sender, EventArgs e) {
    SqlConnection sqlconnectionPubs = new
    SqlConnection("server=(local)\NetSDK;uid=QSUser;pwd=QSPassword;database=p
s");

    SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter("select title,
note price from titles where type='business'", sqlconnectionPubs);

    DataSet datasetTitles = new DataSet();
    sqldataadapterTitles.Fill(datasetTitles, "titles");

    datagridTitles.DataSource=datasetTitles.Tables["titles"].DefaultView;
    datagridTitles.DataBind();
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
```



**10** Create an error on the page by changing the SQL statement to an invalid SQL statement.

**11** Save the page and request it from the Web server.

The error-handling page appears because of the error on the page.

# HANDLE ERRORS PROGRAMMATICALLY

You can use the `Page_Error` event along with enabling custom handling to handle errors on your individual ASP.NET pages. On each ASP.NET page, you can use the `Page_Error` event handler to trap errors on a page and run code to properly respond to the error.

Handling errors programmatically on a page starts with putting an event handler in the server-side code for the page. You can use the `Page_Error` event to send an error message to the user or to check for a specific error to handle that error. The error details are available through the `Server.GetLastError` method. This method returns the `Exception` object that was created for the error. The `Exception`

object is a rich structure that contains detailed information about the trapped error. For example, the `Exception.Source` property contains the name of the application or the object that causes the error. The `Exception.StackTrace` property helps you identify the location in the code where the error occurs, and the `Exception.Message` property gives you error message text.

When you are done responding to the error, you need to then clear the error by using the `Server.ClearError` method. This is done to ensure the error is not bubbled up to any other error handling mechanisms on the site (like the custom error handling in the `web.config` file).

## HANDLE ERRORS PROGRAMMATICALLY

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Page_Error(Object sender, EventArgs e) {
}
protected void Page_Load(Object sender, EventArgs e) {
    SqlConnection sqlconnectionPubs = new
    SqlConnection("server=(local)\\NetSDK;uid=QSPass;pwd=QSPass;database=pubs");

    SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter("select title,
    notes, price from titles where type='business'", sqlconnectionPubs);

    DataSet datasetTitles = new DataSet();
    sqldataadapterTitles.Fill(datasetTitles, "titles");

    datagridTitles.DataSource=datasetTitles.Tables["titles"].DefaultView;
    datagridTitles.DataBind();
}
</SCRIPT>

```

**1** Open `DatagridTemplate.aspx` from the Code Templates directory.

**2** Add the `Page_Error` event handler to the page.

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Page_Error(Object sender, EventArgs e) {
    String stringMessage = "<HTML><FONT FACE =\"Verdana\"><H3>Handle
    Errors Programmatically</H3>"
    + "There was an error processing this page."
    + "</FONT></HTML>";
    Response.Write(stringMessage);
    Server.ClearError();
}
protected void Page_Load(Object sender, EventArgs e) {
    SqlConnection sqlconnectionPubs = new
    SqlConnection("server=(local)\\NetSDK;uid=QSPass;pwd=QSPass;database=pubs");

    SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter("select title,
    notes, price from titles where type='business'", sqlconnectionPubs);

    DataSet datasetTitles = new DataSet();
    sqldataadapterTitles.Fill(datasetTitles, "titles");
}
}
</SCRIPT>

```

**3** Add a string variable to create an HTML message for informing the user that an error has occurred on the page.

**4** Write the string to the Web browser using the `Response` object.

**5** Clear the error using the `Server.ClearError` method.

**Extra**

Because you are on the same page in which the error occurred, you can print out a number of details about the error.

**Example:**

```
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Page_Error(Object sender, EventArgs e) {
    String stringMessage = "<HTML><FONT FACE =\"Verdana\">"
        + "<H3>Handle Errors Programmatically</H3>"
        + "There was an error processing this page."
        + "<P/>Here is the error information:<P/><PRE>" +
Server.GetLastError().ToString()
        + "</PRE></FONT></HTML>";
    Response.Write(stringMessage);
    Server.ClearError();
}
</SCRIPT>
```

```
File Edit Format Help
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Page_Error(Object sender, EventArgs e) {
    String stringMessage = "<HTML><FONT FACE =\"Verdana\"><H3>Handle
Errors Programmatically</H3>"
    + "There was an error processing this page."
    + "</FONT></HTML>";
    Response.Write(stringMessage);
    Server.ClearError();
}

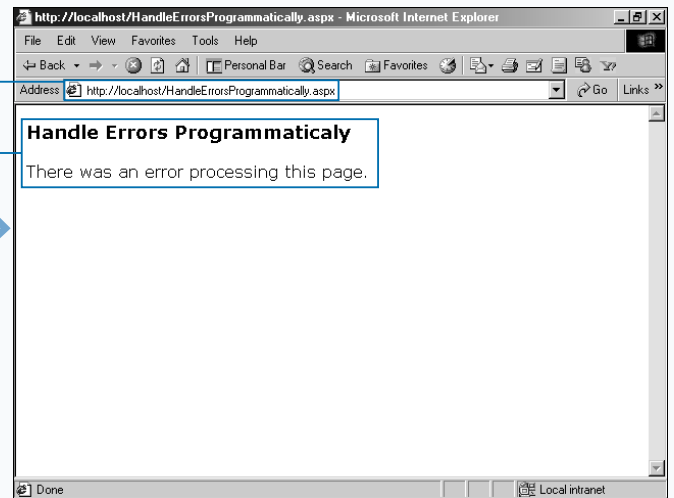
protected void Page_Load(Object sender, EventArgs e) {
    SqlConnection sqlconnectionPubs = new
SqlConnection("server=(local)\NetSDK;uid=QUser,pwd=QSPassword;database=pub
s");

    SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter("select title,
note: price from titles where type='business'", sqlconnectionPubs);

    DataSet datasetTitles = new DataSet();
    sqldataadapterTitles.Fill(datasetTitles);
}
```

**6** Create an error on the page by changing the SQL statement to an invalid SQL statement.

**7** Save the page and request it from the Web server.



You remain on the same page and an error message appears.

# USE A PAGE-LEVEL TRACE

You can use page tracing on your individual ASP.NET pages to get information about the page request when attempting to debug your site. Tracing can be set on the page level with the `@Page` directive. To trace an ASP.NET Web page, you need to add `<%@ Page-Trace="true" %>` to the top of the page. When the page is requested, the trace information appears.

With traces you can inspect the common collection of the `HttpRequest` and execution flow (timing and call stack). The *Trace Information* section displays the different functions and their associated execution times. The *Control Tree* section shows detailed information about the use of controls and control

hierarchy for the page. The *Cookies Collection* section displays all the cookies sent in the request. The *Headers Collection* section shows the name value pairs sent in the header section of the request. The *Server Variables* section displays information about the server, including security and configuration information.

The trace information for a page appears at the bottom of the page. You can add your own trace information to page level traces. The trace information is available to you through the `TraceContext` object. This object can be accessed by using the `Trace` property of a `Page` or through the `HttpContext`.

## USE A PAGE-LEVEL TRACE

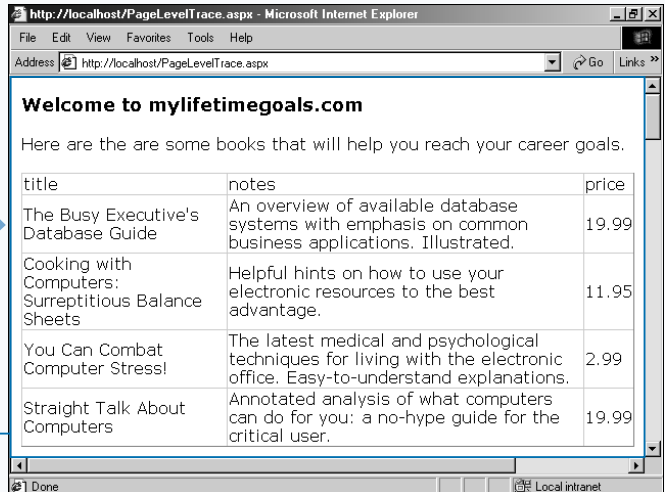
```

Untitled - Notepad
File Edit Format Help
<%@ Page Trace="true" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
protected void Page_Load(Object sender, EventArgs e) {
    SqlConnection sqlconnectionPubs = new
    SqlConnection("server=(local)\NetSDK;uid=QSUser;pwd=QSPassw
    s");

    SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter(
    notes, price from titles where type='business'", sqlconnectionPubs);

    DataSet datasetTitles = new DataSet();
    sqldataadapterTitles.Fill(datasetTitles, "titles");

    datagridTitles.DataSource=datasetTitles.Tables["titles"];
  
```



**1** Open `DatagridTemplate.aspx` from the Code Templates directory.

**2** Turn on tracing for the page by setting the `Trace="true"` attribute for the `@Page` directive.

**3** Save the page and request it from the Web server.

**4** The page contents appear.

**4** Scroll down the page until you get to the Request Details.



**Apply It**

You can write to the Trace Information from within your ASP.NET Web page to track significant sections of code. This is useful not only for outputting values at certain times, but also for seeing how long it takes for something to execute. For the full version of the code refer to PageLevelTrace\_ai.aspx.

**TYPE THIS:**

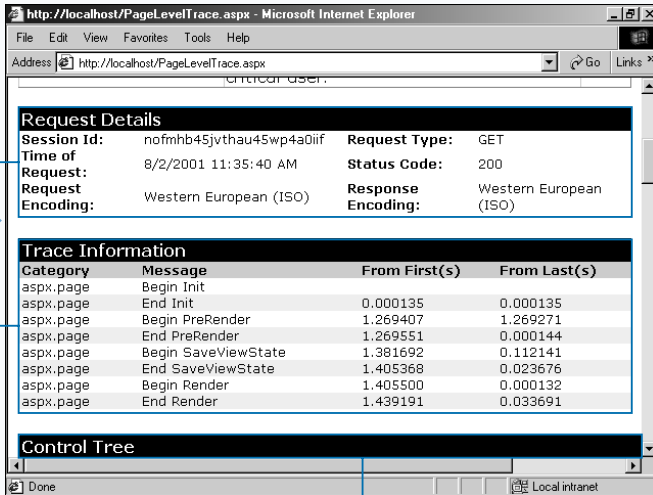
```
protected void Page_Load(Object sender, EventArgs e) {
    SqlConnection sqlconnectionPubs = new SqlConnection
        ("server=(local)\\NetSDK;uid=QSUser;pwd=QSPassword;database=pubs");
    SqlDataAdapter sqldataadapterTitles = new SqlDataAdapter
        ("select title, notes, price from titles "
        + "where type='business'", sqlconnectionPubs);
    DataSet datasetTitles = new DataSet();
    sqldataadapterTitles.Fill(datasetTitles, "titles");

    datagridTitles.DataSource=datasetTitles.Tables["titles"].DefaultView;
    Trace.Write("DataBind", "About to bind the datagrid.");
    datagridTitles.DataBind();
    Trace.Write("DataBind", "Done binding the datagrid.");
}
```



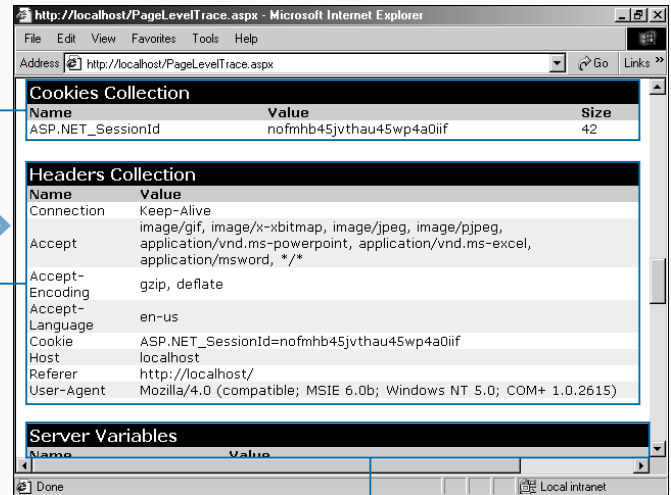
**RESULTS:**

Trace information that includes details on the start and completion of the datagrid binding.



- The Request Details appear.
- The Trace Information appears.

- The Control Tree appears.
- 5 Scroll down the page until you get to the Cookies Collection.



- The Cookies Collection appears.
- The Headers Collection appears.

- The Server Variables appear.

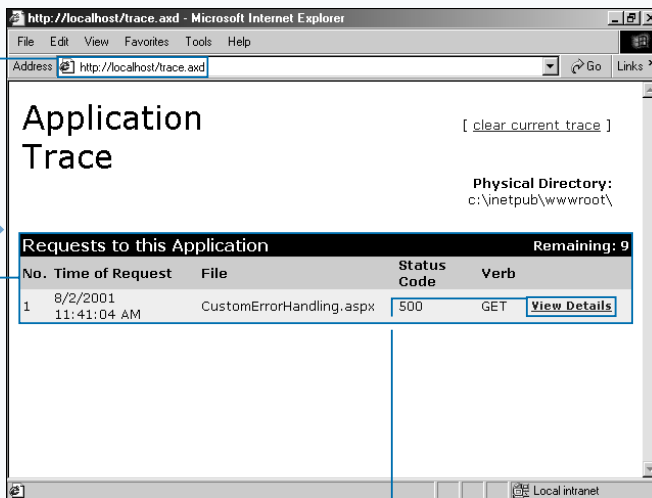
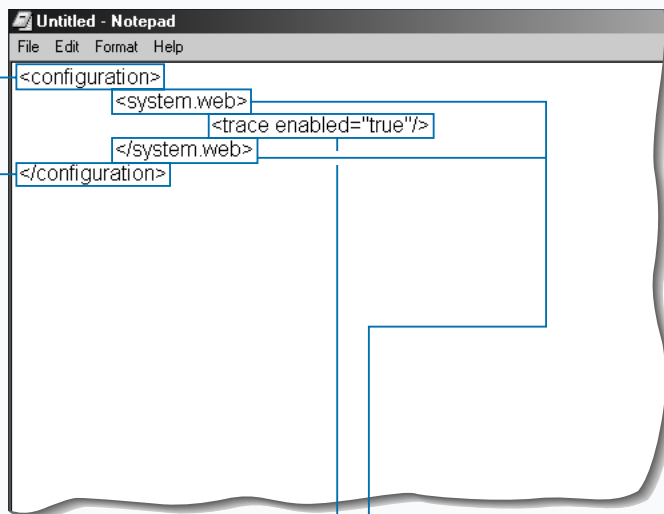
# USE AN APPLICATION-LEVEL TRACE

You can use application-level tracing to view details on a series of requests made to your ASP.NET Web application. Application-level tracing is part of your Web configuration file (`web.config`).

To enable application-level traces, you need to add the `trace` element under the `<system.web>` tag. For application traces to work properly, you need the `web.config` file at the root directory of your ASP.NET application. Therefore, the `web.config` file must be either in its own Web site or virtual directory that is configured as an application (see page 10 for further information on setting up Web sites and virtual directories).

After your site is configured for application tracing, all subsequent requests will be collected in a trace log. When you are ready to view these traces, you request a special file called `trace.axd` from the root directory. The `trace.axd` is not a physical file on your hard drive. When the `trace.axd` is requested in a URL, it will have the Web server generate a page that displays a master list of all the captured traces. From this master list, you can click the “View Details” hyperlink on the last column to see the details of the request. The details are very similar to what you would find on a page-level trace.

## USE AN APPLICATION-LEVEL TRACE



**1** Open a new document in your text editor.

**2** Start a `web.config` file by adding `<configuration>` tags.

**3** Add `<system.web>` tags.

**4** Add a `<trace>` tag and set the `enabled` attribute.

**5** Save the file as `web.config` to the Web site.

**6** Request the `Trace.axd` page in the root directory for the application.

Recent requests are displayed. You may need to open another instance of your Web browser and request some of the other files in the directory to see requests in the trace.

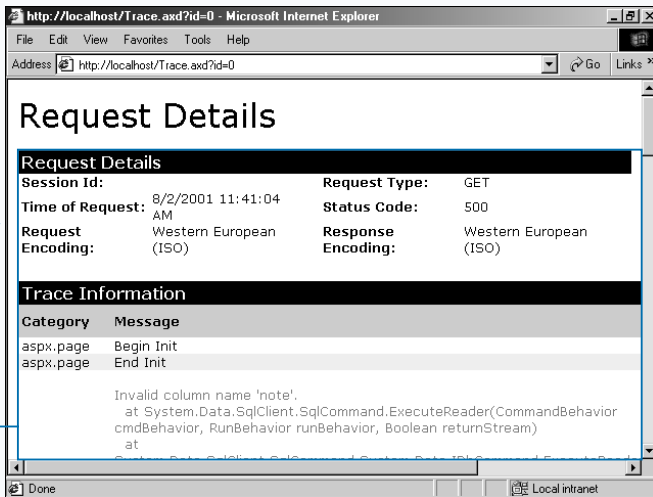
**7** Click the View Details link to see the details for a specific request.

**Extra**

You can fine tune the storage of your trace information with the attributes on the trace element. The requestLimit is the number of requests to trace. The default for this is 10 requests. You can specify whether to have individual pages output trace information by setting the pageOutput = "true". You can also sort the trace information by category, instead of time, by specifying traceMode="SortByCategory".

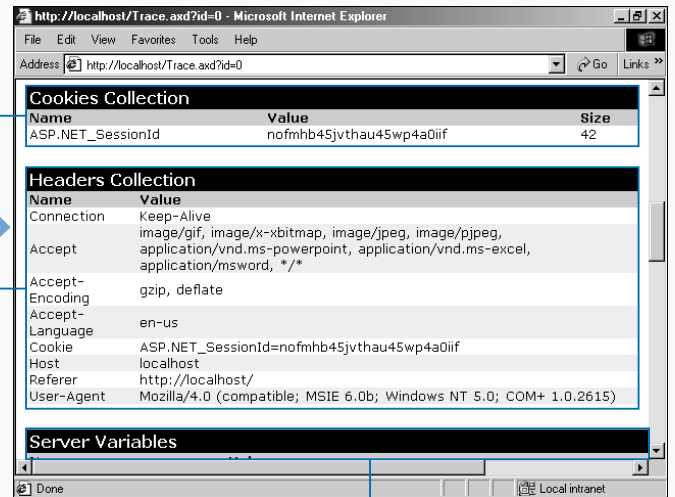
**Example:**

```
<configuration>
  <system.web>
    <customErrors defaultRedirect="error.aspx" mode="on" />
    <trace enabled="true" requestLimit="50" pageOutput="true"
      traceMode="SortByCategory" />
  </system.web>
</configuration>
```



■ Details appear for the request selected, including the Trace Information.

**8** Scroll down the page until you get to the Cookies Collection.



■ The Cookies Collection appears.

■ The Headers Collection appears.

■ The Server Variables appear.

# USING WINDOWS AUTHENTICATION

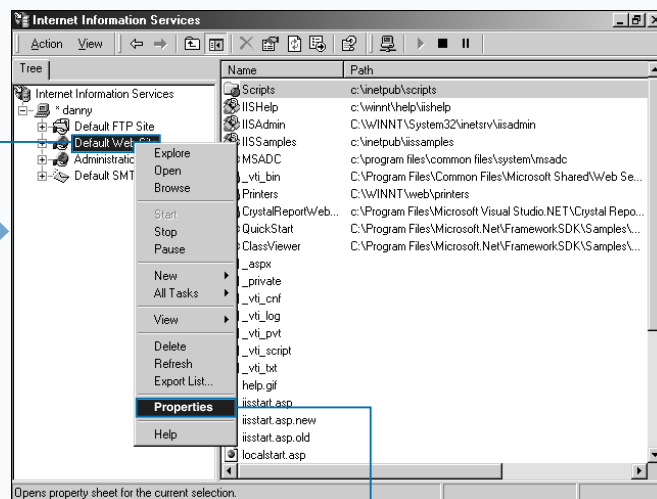
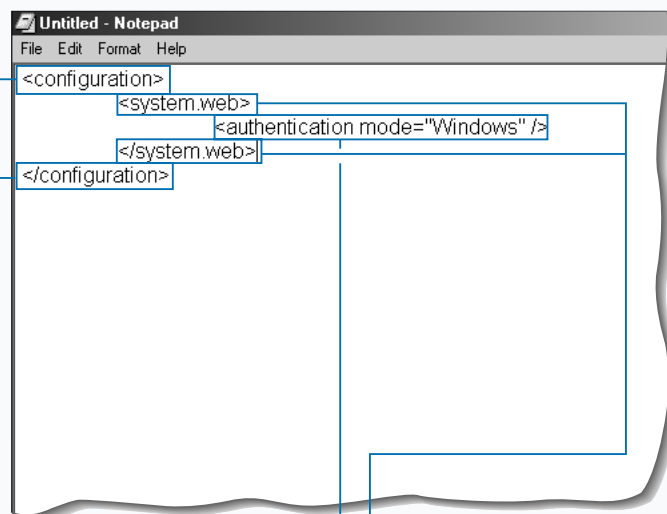
You can use Windows Authentication for securing access to your ASP.NET application. To secure your applications, you can use Windows Authentication in conjunction with the IIS Integrated Windows Authentication feature so that ASP.NET will attempt to use the browser's security context to authenticate the user.

Alternatively, you can use Windows Authentication with Basic Authentication if you want to support a wider range of browser types. IIS's Integrated Windows Authentication works only with Microsoft Internet Explorer. If you are using Basic Authentication, you need to be aware that passwords are sent over the wire in clear text. Consequently, you

should only use Basic Authentication over a Secured Sockets Layer (SSL) via HTTPS.

To set up Windows Authentication, you need to add a section to the `web.config` file. In the `<system.web>` section of the `web.config` file, you can add an `<authentication>` section and set the mode attribute to `Windows`. Additionally, the directory where you have code that uses Windows Authentication needs to be run as an application. You can accomplish this by setting directory properties in Internet Services Manager. Also, to force Windows Authentication, be sure to turn off anonymous access to the application.

## USING WINDOWS AUTHENTICATION



1 Open a new document in your text editor.

2 Add the `<configuration>` start and end tags.

3 Add the `<system.web>` start and end tags.

4 Add an `<authentication>` tag and set an attribute named `mode` equal to `Windows`.

5 Save the file as `web.config`.

6 Open the Internet Services Manager and expand the tree until you get to the directory where you want to save your code for this task.

7 Right-click the directory and choose `Properties`.

## Apply It

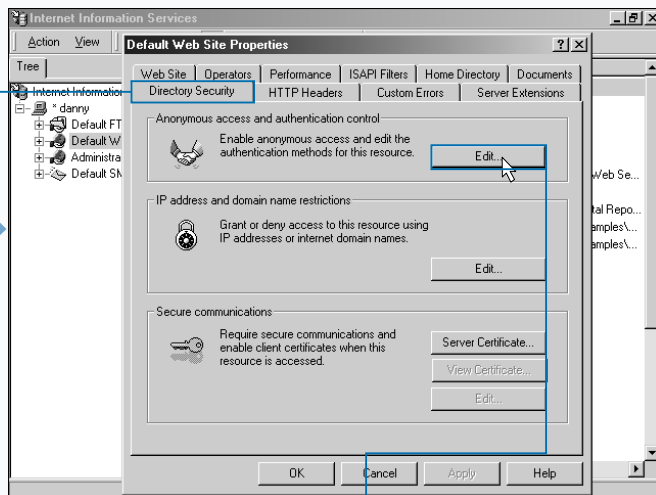
You can check to see if users are authenticated and dynamically update controls to display a message to the users based on whether they are authenticated or not. To get the full code sample, see the `WindowsAuthentication_ai.aspx` file companion CD-ROM.

### TYPE THIS:

```
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Page_Load(Object Src, EventArgs E ) {
    Boolean booleanIsAuthenticated = User.Identity.IsAuthenticated;
    if (booleanIsAuthenticated == true) {
        labelUserName.Text = User.Identity.Name;
        labelAuthenticationType.Text = User.Identity.AuthenticationType;
        labelDisplayUserName.Visible = true;
        labelDisplayAuthenticationType.Visible = true;
        labelDisplayAuthentication.Visible = false;
    }
    else {
        labelDisplayUserName.Visible = false;
        labelDisplayAuthenticationType.Visible = false;
        labelDisplayAuthentication.Visible = true;
    }
}
</SCRIPT>
```

### RESULT:

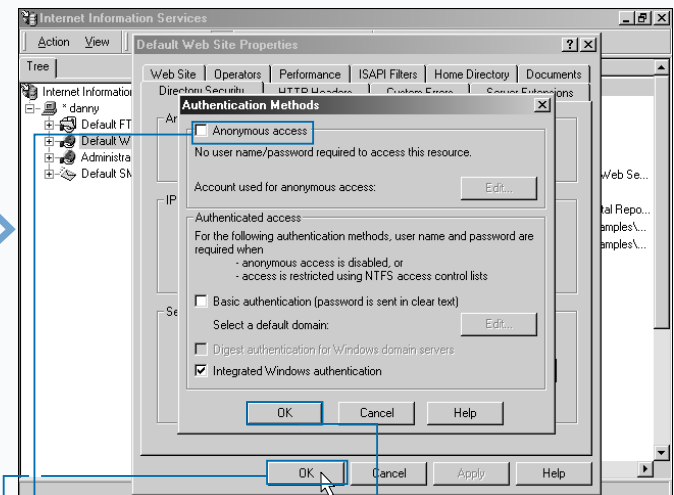
Welcome to [www.mylifetimegoals.com](http://www.mylifetimegoals.com)  
 You are authenticated as TAR-DEV-LAPTOP\Administrator.  
 You are authenticated using NTLM.



8 The Properties dialog box opens.

8 Click the Directory Security tab.

9 Click the Edit button to open the Authentication Methods dialog box.



10 Click the Anonymous access check box to turn off access for the application.

11 Click OK to accept the changes and close the Authentication Methods dialog box.

12 Click OK to close the Properties dialog box.

CONTINUED

# USING WINDOWS AUTHENTICATION

You can use Windows Authentication to manage authentication with user accounts that are stored in your Windows 2000 domain. Using Windows Authentication assumes that the user accessing your site has a user account on the Windows 2000 Server domain that is running the Web server.

For users to request a resource on a Web site, they must be mapped to a valid user account on that Web server. Most publicly available sites on the World Wide Web do not want to create a Windows 2000 account for every user of the site. This would be an administrative nightmare, so sites are typically configured to run all users under the same account.

Administrators do this by enabling *anonymous* access. Anonymous access will map all users that access a Web server to one account that you specify in the Internet Services Manager.

Administrators must also manage another security concept — *impersonation*. After a user has access to the Web server, he or she has to make requests on behalf of the user that requested a URL. You have the ability to impersonate another user if your Web application needs to run under one account for all users. You can configure impersonation in the `web.config` file with the `identity` element under the `<system.web>` tag.

## USING WINDOWS AUTHENTICATION (CONTINUED)

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to www.mylifetimegoals.com</H3>

You are authenticated as <ASP:LABEL ID="labelUserName" RUNAT="Server"/>. <BR/>
You are authenticated using <ASP:LABEL ID="labelAuthenticationType"
RUNAT="Server"/>. <BR/>

</FONT>
</BODY>
</HTML>

```

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Page_Load(Object Src, EventArgs E) {
}
</SCRIPT>|
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to www.mylifetimegoals.com</H3>

You are authenticated as <ASP:LABEL ID="labelUserName" RUNAT="Server"/>. <BR/>
You are authenticated using <ASP:LABEL ID="labelAuthenticationType"
RUNAT="Server"/>. <BR/>

</FONT>
</BODY>
</HTML>

```

**13** Open the `GenericTemplate.aspx` template from the Code Templates directory.

**14** Add a heading for the page.

**15** Add a message to the user about their authentication and use a label control to hold the value.

**16** Add a message to the user about how they are authenticated as and use a label control to hold the value.

**17** Add the `Page_Load` event handler to the page by using the `<SCRIPT>` tags.

**Extra**

When securing Web applications, you deal with authentication, authorization, and impersonation.

*Authentication* is the process of identifying if you are a configured user of the system. Authentication occurs after the user provides a name/password pair that they enter when logging on to the site. This name/password pair is also called the *user credentials*.

After you have authenticated a user, you need a way to determine the appropriate access rights of the user to read, modify, delete, and so on, resources on your Web site. This is known as *user authorization*.

Sometimes the user is mapped over to an account that is shared by multiple users. This is called *impersonation* and is used for setting up anonymous access to a Web application in IIS 5.0.

Be cautious when configuring impersonation, because configuration requires you to insert the password in the text of the `web.config` file. (Passwords in a text file are not very secure.)

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Page_Load(Object Src, EventArgs E) {
    labelUserName.Text = User.Identity.Name;
    labelAuthenticationType.Text = User.Identity.AuthenticationType;
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3>Welcome to www.mylifetimegoals.com</H3>
You are authenticated as <ASP:LABEL ID="labelUserName" RUNAT="Server"/>. <BR/>
You are authenticated using <ASP:LABEL ID="labelAuthenticationType"
RUNAT="Server"/>. <BR/>
</FONT>
</BODY>
</HTML>
  
```

**18** Set the values for the labels including the username and the authentication type.

```

http://localhost/WindowsAuthentication.aspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Home Personal Bar Search Favorites
Address http://localhost/WindowsAuthentication.aspx Go Links
Welcome to www.mylifetimegoals.com
You are authenticated as DANNY\Administrator.
You are authenticated using NTLM.
Done Local intranet
  
```

**19** Save the file and request it from the Web server.

A message appears showing the username and authentication type.

# USING FORMS AUTHENTICATION

You can build a custom login page with Forms Authentication for securing your ASP.NET applications. ASP.NET Forms Authentication is not the most secure option, but if you cannot use Integrated Windows Authentication or do not want to use the Windows Logon dialog box, it is the best alternative.

Forms Authentication uses cookies to indicate whether the user is authenticated. When users access a resource without the cookie present, they are redirected to a predetermined custom login page that collects authentication information. When users submit their user credentials, the page authenticates the user. If authenticated, the Web server sends back an

authentication cookie in the header. This cookie is passed in the request header in future requests to allow users to bypass the login page on subsequent page request. The user will have this cookie until the specified timeout occurs.

To set up Forms Authentication, you need to add an `<authentication>` section to your `web.config` file. After you have set up the authentication section, you can use an `<authorization>` section to give specific rights to users (note that `?` represents all anonymous identities, and `*` represents all identities). You also need to set up the directory to run as an application using the Internet Services Manager.

## USING FORMS AUTHENTICATION

```

<configuration>
  <system.web>
    <authentication mode="Forms">
    </authentication>
  </system.web>
</configuration>

```

**1** Open a new document in your text editor.

**2** Add the `<configuration>` start and end tags.

**3** Add the `<system.web>` start and end tags.

**4** Add an `<authentication>` tag and set the mode attribute equal to `Forms`.

```

<configuration>
  <system.web>
    <authentication mode="Forms">
      <forms name=".ASPXUSERDEMO"
        loginUrl="FormsAuthenticationLogin.aspx" protection="all" timeout="60" />
    </authentication>
    <authorization>
      <deny users="?" />
    </authorization>
  </system.web>
</configuration>

```

**5** Add a `<forms>` tag and set the name equal attribute to a unique name, a `loginURL` attribute equal to the name of your login page, a `protection` attribute equal to the value of `all`, and a `timeout` attribute equal to `60`.

**6** Add a set of `<authorization>` start and end tags.

**7** Add a `<deny/>` tag with the `users` attribute set to `?`.

**8** Save the file as `web.config`.



## Apply It

When working with Forms-based authentication, a login page must collect user credentials and authenticate (maybe checking a database against the supplied credentials). If the users pass authentication, you can redirect them back to the originally requested page. To get the full code sample, see `FormsAuthenticationLogin_ai.aspx` file on the companion CD-ROM.

### TYPE THIS:

```
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void SubmitBtn_Click(object Source, EventArgs e) {
    if (inputPassword.Text == "goals") {
        FormsAuthentication.RedirectFromLoginPage(inputName.Text, false);
    }
    else {
        labelMessage.Text="That password is not correct.";
    }
}
</SCRIPT>
```

### RESULT:

If you request another page in the site (test with `FormsAuthenticationDefault.aspx` page), you will get this custom login page. After you supply credentials (where password = "goals"), you will be sent to the original page that you requested.

```
Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3>Welcome to www.mylifetimegoals.com <ASP:LABEL ID="labelUserName"
RUNAT="Server"/>|</H3>
</FONT>
</BODY>
</HTML>
```

**9** Open the `GenericTemplate.aspx` template from the Code Templates directory.

**10** Add a heading for the page that contains a label control for displaying the user name.

```
Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Page_Load(object Source, EventArgs e) {
    labelUserName.Text = User.Identity.Name;
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3>Welcome to www.mylifetimegoals.com <ASP:LABEL ID="labelUserName"
RUNAT="Server"/>|</H3>
</FONT>
</BODY>
</HTML>
```

**11** Create the `Page_Load` event handler to set the user name for the label control.

**12** Save the file as `FormsAuthenticationDefault.aspx`.

CONTINUED 

# USING FORMS AUTHENTICATION

If you use Forms Authentication, you can use user data stores other than Windows 2000 domain accounts for determining valid users. To set up Forms Authentication, you need to create a login page that authenticates the user. ASP.NET uses the page specified in the `loginUrl` attribute of the forms element found under the `<authentication>` section. At a minimum, this page requires a place for the user to enter a user name and password. You could include some other special credentials that are part of identifying a user, such as the company name. After you retrieve all necessary credentials, you can check them against the store of your user data. This could be a SQL database, Active Directory, or some

other user data store. After users log into this page with your credentials, they are redirected to the original page. This redirection is not automatic. It is programmed into the function that handles the submit of the login page with the use of the `FormsAuthentication.RedirectFromLoginPage` method.

The form used to collect the user credentials contains sensitive information and should not be sent over an unencrypted line. To protect your user credentials, you want to put this form in a protected directory where Secured Sockets Layer (SSL) is configured. This will send this page over HTTPS instead of HTTP.

## USING FORMS AUTHENTICATION (CONTINUED)

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Please login to the secured area. You can use "goals" as a guest password.

<FORM RUNAT="Server">
  Enter Name: <ASP:TEXTBOX ID="inputName" TEXTMODE="SingleLine" TEXT=""
  WIDTH="200px" RUNAT="Server"/>
  <BR/>
  Enter Password: <ASP:TEXTBOX ID="inputPassword" TEXTMODE="Password"
  TEXT="" WIDTH="200px" RUNAT="Server"/>
  <P/>
  <ASP:BUTTON OnClick="SubmitBtn_Click" TEXT="Submit" RUNAT="Server"/>
  <P/>
  <ASP:LABEL ID="labelMessage" style="color:red" RUNAT="Server"/>
</FORM>

</FONT>
  
```

**13** Open the `GenericTemplate.aspx` template from the Code Templates directory.

**14** Add a heading for the page and a message about login.

**15** Add a form control to the page.

**16** Add a text box for the user name and password, along with a submit button and a label for displaying messages.

```

Untitled - Notepad
File Edit Format Help
<%@ Import Namespace="System.Web.Security" %>
<HTML>
<HEAD>

<SCRIPT LANGUAGE="C#" RUNAT="Server">
void SubmitBtn_Click(object Source, EventArgs e) {
  if (inputPassword.Text == "goals") {
    FormsAuthentication.RedirectFromLoginPage(inputName.Text, false);
  }
  else {
    labelMessage.Text="That password is not correct.";
  }
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Please login to the secured area. You can use "goals" as a guest password.

<FORM RUNAT="Server">
  Enter Name: <ASP:TEXTBOX ID="inputName" TEXTMODE="SingleLine" TEXT=""
  WIDTH="200px" RUNAT="Server"/>
  <BR/>
  Enter Password: <ASP:TEXTBOX ID="inputPassword" TEXTMODE="Password"
  TEXT="" WIDTH="200px" RUNAT="Server"/>
  <P/>
  <ASP:BUTTON OnClick="SubmitBtn_Click" TEXT="Submit" RUNAT="Server"/>
  <P/>
  <ASP:LABEL ID="labelMessage" style="color:red" RUNAT="Server"/>
</FORM>

</FONT>
  
```

**17** Add an alias to the `System.Web.Security` namespace using `@Import`.

**18** Create the `SubmitBtn_Click` event handler.

**19** Check the password field for the correct input and use the `RedirectFromLoginPage` to forward on the user.

**20** If the user did not enter correct input, set an appropriate message for the user.

## Apply It

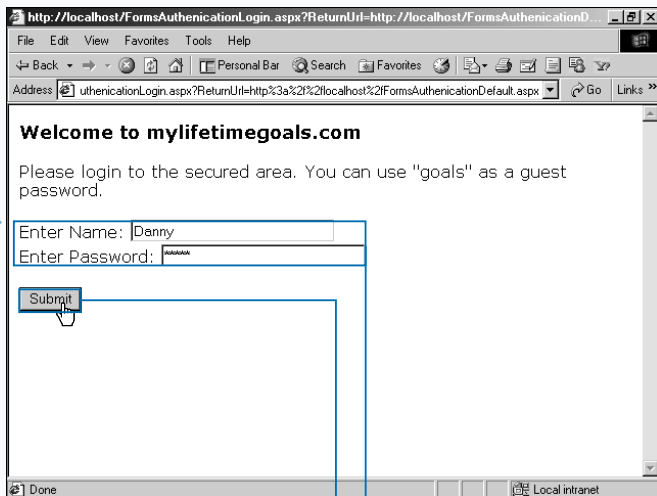
When collecting user credentials, you can validate server controls to ensure that the user enters all required fields before the validation check is performed.

### TYPE THIS:

```
<BODY>
<FONT FACE ="Verdana"><H3>Welcome to mylifetimegoals.com</H3>
Please login to the secured area. You can use "goals" as a guest password.
<FORM RUNAT="Server">
  Enter Name: <ASP:TEXTBOX ID="inputName"
    TEXTMODE="SingleLine" TEXT="" WIDTH="200px"
    RUNAT="Server" />
  <ASP:REQUIREDFIELDVALIDATOR CONTROLTOVALIDATE="inputName"
    DISPLAY="Static" ERRORMESSAGE="Please enter your name."
    RUNAT="Server" />
  <BR/>
  Enter Password: <ASP:TEXTBOX ID="inputPassword"
    TEXTMODE="Password" TEXT="" WIDTH="200px"
    RUNAT="Server" />
  <ASP:REQUIREDFIELDVALIDATOR CONTROLTOVALIDATE="inputPassword"
    DISPLAY="Static" ERRORMESSAGE="Please enter a password." RUNAT="Server" />
  <P/>
  <ASP:BUTTON OnClick="SubmitBtn_Click" TEXT="Submit" RUNAT="Server" />
  <P/>
  <ASP:LABEL ID="labelMessage" style="color:red" RUNAT="Server" />
</FORM>
</FONT>
</BODY>
```

### RESULT:

If you request another page in the site (test with FormsAuthenticationDefault.aspx page), you will get this custom login page. After you supply credentials (where password = "goals"), you will be sent to the original page that you requested.

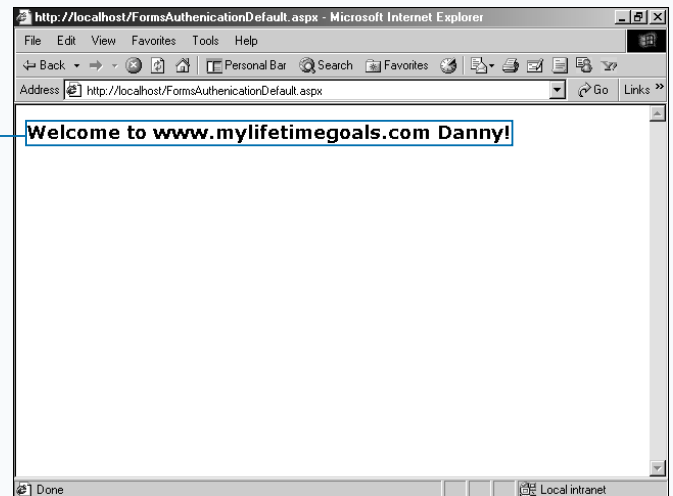


**21** Save the file as **FormsAuthenticationLogin.aspx** and request the **FormsAuthenticationDefault.aspx** file from the Web server.

The **FormsAuthenticationLogin.aspx** page appears.

**22** Type a name and goal for the password.

**23** Click the Submit button.



The **FormsAuthenticationDefault.aspx** page appears, and the message is personalized by using your user name.

# AUTHORIZE USERS

ASP.NET gives you a convenient means to authorize and deny access to resources in your ASP.NET application. You set up authorization in the `<authorization>` section of the `web.config` file. You can use the `<deny/>` tag to deny specific users access and use the `<allow/>` tag to authorize specific users. Note the use of `?`, which is used to represent anonymous identities, and the use of `*`, which represents all entities. You can use commas to delimit users when you wish to specify multiple users in a single tag. You can also specify users in specific domains if you are using IIS's Integrated Windows Authentication by prefixing the domain name (for example, `Domain\UserName`).

If you want to be less granular with authorization, you can allow or deny Windows 2000 domain groups. This can be done with the `roles` attribute of the `allow` element. You can also control the actions that a user is allowed to perform. This is done with the `verb` attribute on the `allow` element. The verbs that we can control are `GET`, `HEAD`, and `POST`. If you do not want a specific user to post data to the Web server but only request Web pages for viewing, you can specify the following:

```
<allow verb="GET" users="*" />
```

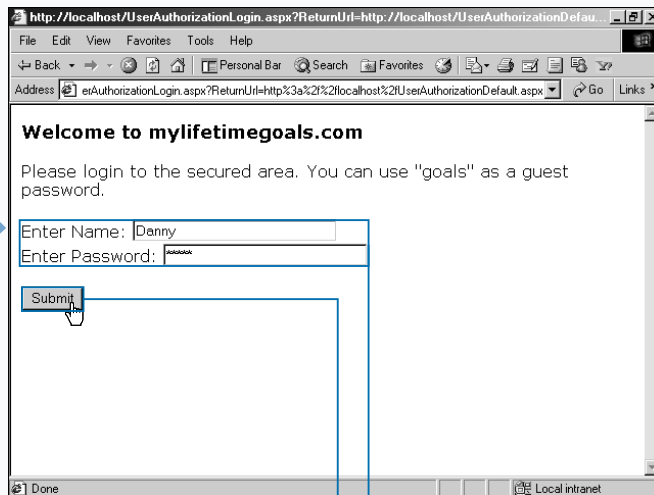
```
<deny verb="POST" users="Linda" />
```

## AUTHORIZE USERS

```
web.config - Notepad
File Edit Format Help
<configuration>
  <system.web>
    <authentication mode="Forms">
      <forms name="ASPXUSERDEMO"
loginUrl="UserAuthorizationLogin.aspx" protection="all" timeout="60" />
    </authentication>
    <authorization>
      <deny users="?" />
      <deny users="Danny" />
      <allow users="Tommy,Deanna,Bobby" />
    </authorization>
  </system.web>
</configuration>
```

- 1 Open the `web.config` template file from the Code Templates directory.
- 2 Add another `<deny/>` tag between the `<authorization>` tags; set `users` attribute to Danny.

- 3 Add an `<allow/>` tag and set the `users` attribute to Tommy, Deanna, Bobby.
- 4 Save the `web.config` file.



- 5 Copy the files `UserAuthorizationDefault.aspx` and `UserAuthorizationLogin.aspx` from the CD-ROM to the Web site and request `UserAuthorizationDefault.aspx` from the Web server.

- 6 Type an unauthorized user's name and goals for the password.
- 7 Click the Submit button.

## Apply It

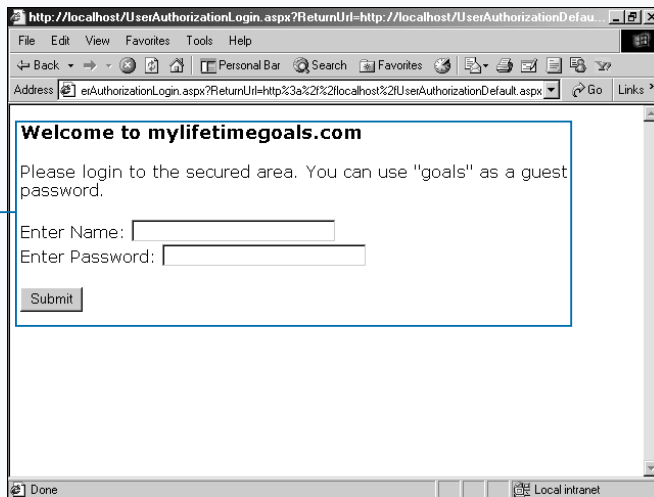
With Forms-based authentication, you can log users out by removing their authentication cookie.

### TYPE THIS:

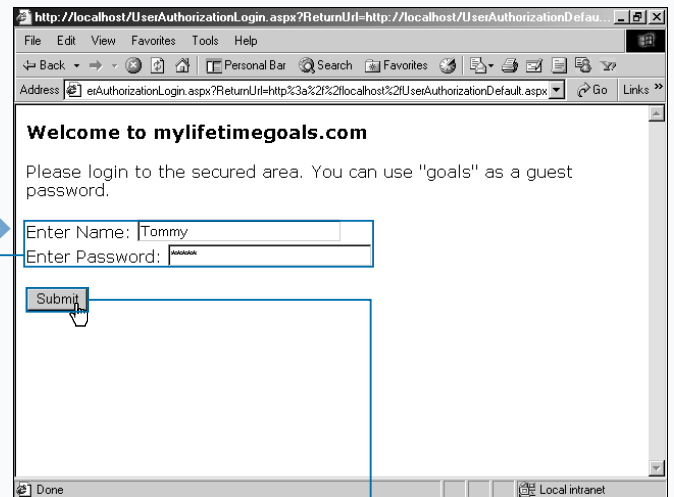
```
<%@ Import Namespace="System.Web.Security" %>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void Page_Load(object Source, EventArgs e) {
    labelUserName.Text = User.Identity.Name;}
void Button_OnClick(Object sender, EventArgs E) {
    FormsAuthentication.SignOut();
    Response.Redirect("UserAuthorizationLogin.aspx");}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE="Verdana">
<H3>Welcome to www.mylifetimegoals.com <ASP:LABEL ID=
"labelUserName" RUNAT="Server"/>!</H3>
<FORM RUNAT="Server">
<ASP:BUTTON ID="buttonSignout" TEXT="Signout" onClick=
"Button_OnClick" RUNAT="Server"/>
</FORM>
</FONT>
</BODY>
</HTML>
```

### RESULT:

After logging in, you can click the Signout button to remove your authentication cookie and be redirected to the login page.



8 You are redirected back to the login page because the user is denied access.



8 Enter an authorized name and **goals** for the password.

9 Click the Submit button.  
 Access is given to view the default page.

# SET UP ENCODING

You can use encoding to create a site that supports multiple languages. *Encoding* refers to the way that the data within your file is stored. Certain languages, such as Japanese, have large character sets and therefore require an encoding type that would support all of the characters in the language.

Encoding is important because it determines how the data from your Web server is sent to and from the Web browser, along with how the files are stored on your Web server. *Response encoding* refers to the way the responses are sent to the Web browser, whereas *request encoding* refers to the way the Web server handles sent requests. *File encoding* pertains to how files are stored on the Web server.

*UCS Transformation Format (UTF-8)* is an encoding format that supports 8-bit form. This encoding supports all Unicode character values, which allow for support of most modern character sets.

Rather than specifying encoding for each page in your application, you can specify this in your `web.config` file for the entire application.

You can specify the response encoding type on a page to be UTF-8 using the `@Page` directive. You can then add multiple languages to the page to accommodate users that speak different languages.

## SET UP ENCODING

```
Untitled - Notepad
File Edit Format Help
<%@Page Language="C#" ResponseEncoding="UTF-8"%>
<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3></H3>

</FONT>
</BODY>
</HTML>
```

**1** Open the `GenericTemplate.aspx` template from the Code Templates directory.

**2** Add the `@Page` directive to the page and set the `ResponseEncoding` attribute to UTF-8, as well as the `Language` attribute to C#.

```
SetUpEncodings.aspx - Notepad
File Edit Format Help
<%@Page Language="C#" ResponseEncoding="UTF-8"%>
<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to www.mylifetimegoals.com!</H3>
<H3>Bienvenido a www.mylifetimegoals.com!</H3>

</FONT>
</BODY>
</HTML>
```

*Note: You will need to install support for languages not currently installed on your computer.*

**3** Add an English heading for the page.

**4** Add a Spanish heading for the page.

**Extra**

You can set up encoding for the entire application in the `web.config` file with the `<globalization>` tag.

The options for the attributes for encoding are as follows:

<code>requestEncoding</code>	Specifies the assumed encoding of each incoming request. The default is <code>us-ascii</code> .
<code>responseEncoding</code>	Sets the content encoding of responses. The default is <code>iso-8859-1</code> .
<code>fileEncoding</code>	Designates the default encoding for <code>.aspx</code> , <code>.asmx</code> , and <code>.asax</code> files.

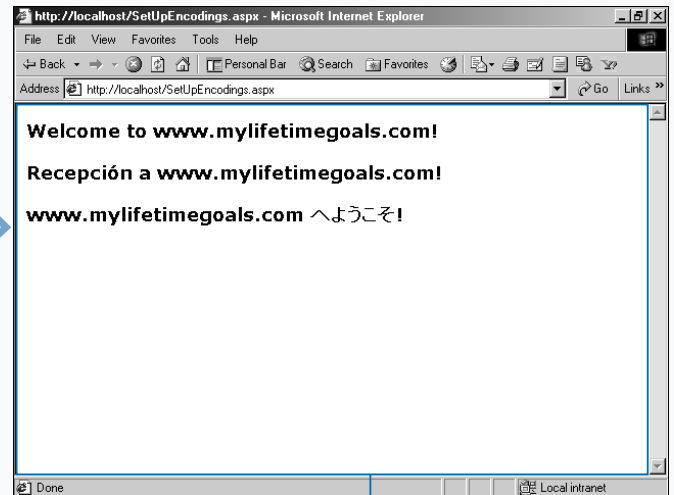
**Example:**

```
<configuration>
  <system.web>
    <globalization " requestEncoding="utf-8" responseEncoding="utf-8" fileEncoding="utf-8" />
  </system.web>
</configuration>
```



```
SetupEncodings.aspx - Notepad
File Edit Format Help
<%@Page Language="C#" ResponseEncoding="UTF-8"%>
<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3>Welcome to www.mylifetimegoals.com!</H3>
<H3>Bienvenido a www.mylifetimegoals.com!</H3>
<H3>www.mylifetimegoals.com 〇〇〇〇!</H3>
</FONT>
</BODY>
</HTML>
```

**5** Add a Japanese heading for the page.



**6** Save the file and request it from the Web server.

The messages in the various languages appear.

# USING CULTUREINFO

You can use the `CultureInfo` class to display localized settings. For example, you can display the date in multiple formats based on the user's preferences. Begin by ensuring that the response's encoding type is appropriate. You can either do this at the Page level or the Application level. See page 262 for more information on setting up encoding. You need to have some way to determine which culture to use. One way to do this is to simply have the user select the culture. Another way is to read this information from the user's Web browser. After you

determine the culture, you can set the current thread's culture and then access the appropriate property.

You can also use the `CultureInfo` class to display the calendar preferences and the native name of the culture.

You can create a drop-down list of cultures so the user can specify a culture. After the user selects a culture, you can use `CultureInfo` and a label control to display the localized time.

## USING CULTUREINFO

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Select your culture below:

<FORM RUNAT="Server">
<P/>
<ASP:DROPDOWNLIST ID="dropdownlistCultures" RUNAT="Server">
  <ASP:LISTITEM VALUE="en-US">English</ASP:LISTITEM>
  <ASP:LISTITEM VALUE="es-ES">Spanish</ASP:LISTITEM>
  <ASP:LISTITEM VALUE="ja-JP">Japanese</ASP:LISTITEM>
</ASP:DROPDOWNLIST>

</FORM>

</FONT>
</BODY>

```

- 1 Open the `GenericTemplate.aspx` template from the Code Templates directory.
- 2 Add a heading to the page.
- 3 Add a message to the user.

```

Untitled - Notepad
File Edit Format Help
<%@Page Language="C#" ResponseEncoding="utf-8" %>
<%@Import Namespace="System.Threading"%>
<%@Import Namespace="System.Globalization"%>
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Select your culture below:

<FORM RUNAT="Server">
<P/>
<ASP:DROPDOWNLIST ID="dropdownlistCultures" RUNAT="Server">
  <ASP:LISTITEM VALUE="en-US">English</ASP:LISTITEM>
  <ASP:LISTITEM VALUE="es-ES">Spanish</ASP:LISTITEM>
  <ASP:LISTITEM VALUE="ja-JP">Japanese</ASP:LISTITEM>
</ASP:DROPDOWNLIST>
<ASP:BUTTON TEXT="Submit" OnClick="SubmitBtn_Click" RUNAT="Server"/>
<P/>
<SPAN ID="spanMessage" RUNAT="Server" />
</FORM>

```

- 4 Add a form control.
- 5 Create a drop-down list that has several languages from which to choose.
- 6 Add a submit button control.
- 7 Add a span control on the form to display messages.
- 8 Add the `@Page` directive to the page and set the `ResponseEncoding` attribute to `UTF-8`, as well as the `Language` attribute to `C#`.
- 9 Add an alias to the `System.Threading` and `System.Globalization`



## Apply It

In addition to displaying the date and time in local format, you can also display a culture-specific calendar that uses the native language for the culture.

### TYPE THIS:

```
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void SubmitBtn_Click(object Source, EventArgs e) {
    CultureInfo cultureinfoLanguage = new
    CultureInfo(dropDownListCultures.SelectedItem.Value);
    Thread.CurrentThread.CurrentCulture = cultureinfoLanguage;
    spanMessage.InnerHtml = "The localized date is " +
    DateTime.Now.ToString("D", CultureInfo.CurrentCulture) +
    ". " + "<BR/> The calendar to use for this culture is " +
    CultureInfo.CurrentCulture.Calendar + ". " +
    "<BR/> The native name for this culture is " +
    CultureInfo.CurrentCulture.NativeName + ". ";
}
</SCRIPT>
```

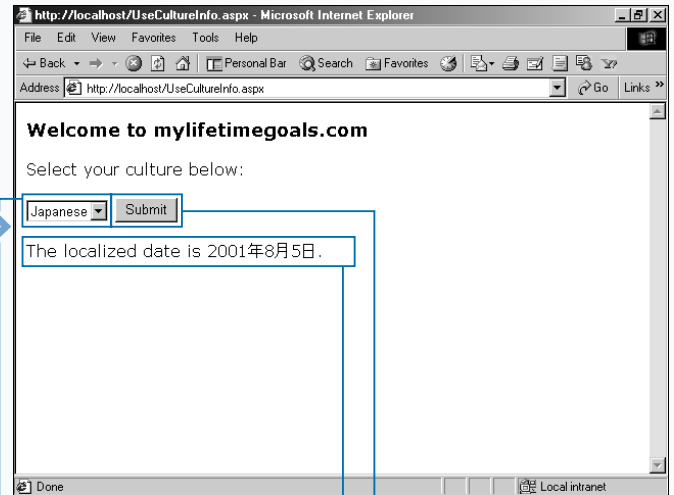
### RESULT:

The localized date is domingo, 05 de agosto de 2001.  
 The calendar to use for this culture is System.Globalization.GregorianCalendar.  
 The native name for this culture is español (España).

```
Untitled - Notepad
File Edit Format Help
<%@Page Language="C#" ResponseEncoding="utf-8" %>
<%@Import Namespace="System.Threading"%>
<%@Import Namespace="System.Globalization"%>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void SubmitBtn_Click(object Source, EventArgs e) {
    CultureInfo cultureinfoLanguage = new
    CultureInfo(dropDownListCultures.SelectedItem.Value);
    Thread.CurrentThread.CurrentCulture = cultureinfoLanguage;
    spanMessage.InnerHtml = "The localized date is " +
    DateTime.Now.ToString("D", CultureInfo.CurrentCulture) + ". ";
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
Select your culture below:
<FORM RUNAT="Server">
<P/>
<ASP:DROPDOWNLIST ID="dropDownListCultures" RUNAT="Server">
```

- 10 Add the `SubmitBtn_Click` function.
- 11 Create a new `CultureInfo` variable based on the item selected in the drop-down list box.

- 12 Set the current culture.
- 13 Set the `InnerHtml` property of the span control to the date based on the current culture.



- 14 Save the file and request it from the Web server.
- 15 Click  to select a culture.

- 16 Click the Submit button. The message appears according to the culture selected.

# USING REGIONINFO

You can use the `RegionInfo` class to show regionalized settings. For example, you can display the local currency based on the user's preferences.

Working with the `RegionInfo` class is similar to working with the `CultureInfo` class, you must ensure that the response's encoding type is set up appropriately. You can either do this at the Page level or the Application level. See page 262 for more information on setting up encoding. You need to determine which region to use for the user. You can simply have the user select the culture, or you can try to determine it from the information sent from the

user's Web browser. After you determine the region, you can then access the property with which you want to work.

You can also use the `RegionInfo` class to determine whether the region uses the metric system and what the Windows Region Name is. See the Apply It! section for the code necessary to do this.

You can create a drop-down list of regions for the user to choose from on a Web page. After the user selects a region, you can use the `RegionInfo` class and a label control to display the regional currency for the user.

## USING REGIONINFO

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Select your region below.

<FORM RUNAT="Server">
<P/>
<ASP:DROPDOWNLIST ID="dropdownlistRegions" RUNAT="Server">
  <ASP:LISTITEM VALUE="US">United States</ASP:LISTITEM>
  <ASP:LISTITEM VALUE="ES">Spain</ASP:LISTITEM>
  <ASP:LISTITEM VALUE="JP">Japan</ASP:LISTITEM>
</ASP:DROPDOWNLIST>

</FORM>

</FONT>
</BODY>
  
```

**1** Open the `GenericTemplate.aspx` template from the Code Templates directory.

**2** Add a heading to the page.

**3** Add a message to the user.

**4** Add a form control.

**5** Create a drop-down list that has several languages from which to choose.

```

Untitled - Notepad
File Edit Format Help
<%@Page Language="C#" ResponseEncoding="utf-8" %>
<%@Import Namespace="System.Threading"%>
<%@Import Namespace="System.Globalization"%>
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Select your region below.

<FORM RUNAT="Server">
<P/>
<ASP:DROPDOWNLIST ID="dropdownlistRegions" RUNAT="Server">
  <ASP:LISTITEM VALUE="US">United States</ASP:LISTITEM>
  <ASP:LISTITEM VALUE="ES">Spain</ASP:LISTITEM>
  <ASP:LISTITEM VALUE="JP">Japan</ASP:LISTITEM>
</ASP:DROPDOWNLIST>
<ASP:BUTTON TEXT="Submit" OnClick="SubmitBtn_Click" RUNAT="Server"/>
<SPAN ID="spanMessage" RUNAT="Server" />
</FORM>
  
```

**6** Add a submit button control.

**7** Add a span control on the form to display messages.

**8** Add the `@Page` directive to the page and set the `ResponseEncoding` attribute to `UTF-8`, as well as the `Language` attribute to `C#`.

**9** Add an alias to the `System.Threading` and `System.Globalization` namespaces.

## Apply It

In addition to determining the local date and time, you can also determine whether the region uses the metric system and what the three-letter Windows Region Name is.

### TYPE THIS:

```
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void SubmitBtn_Click(object Source, EventArgs e) {
    RegionInfo regioninfoLanguage =
    new RegionInfo(dropDownListRegions.SelectedItem.Value);
    String stringMetric = "";
    if (regioninfoLanguage.IsMetric == true)
        stringMetric = "uses";
    else
        stringMetric = "does not use";
    spanMessage.InnerHtml = "The local currency for " +
    regioninfoLanguage.EnglishName + " is " +
    regioninfoLanguage.CurrencySymbol + "." +
    "<BR/> This region " + stringMetric +
    " the metric system." +
    "<BR/> The Windows region name for this region is " +
    regioninfoLanguage.ThreeLetterWindowsRegionName + ".";
}
</SCRIPT>
```

### RESULT:

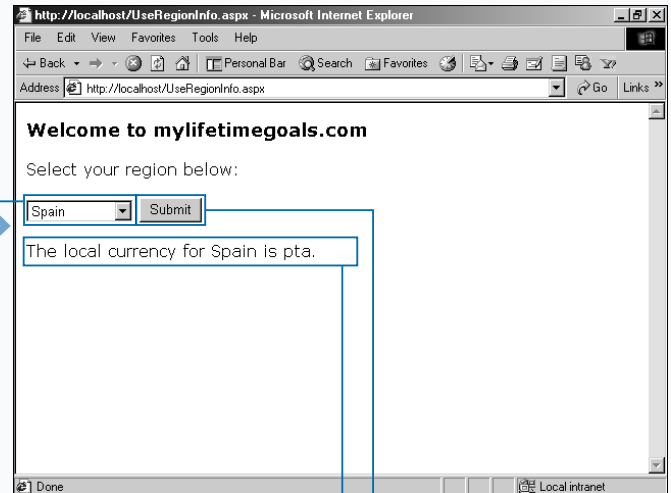
The local currency for United States is \$.  
This region does not use the metric system.  
The Windows region name for this region is USA.

```
Untitled - Notepad
File Edit Format Help
<%@Page Language="C#" ResponseEncoding="utf-8" %>
<%@Import Namespace="System.Threading"%>
<%@Import Namespace="System.Globalization"%>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void SubmitBtn_Click(object Source, EventArgs e) {
    RegionInfo regioninfoLanguage = new
    RegionInfo(dropDownListRegions.SelectedItem.Value);
    spanMessage.InnerHtml = "The local currency for " +
    regioninfoLanguage.EnglishName + " is " + regioninfoLanguage.CurrencySymbol + ".";
}
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3>Welcome to mylifetimegoals.com</H3>
Select your region below:
<FORM RUNAT="Server">
<P>
<ASP:DROPDOWNLIST ID="dropDownListRegions" RUNAT="Server">
<ASP:LISTITEM VALUE="US">United States</ASP:LISTITEM>
```

**10** Add the `SubmitBtn_Click` function.

**11** Create a new `RegionInfo` variable based on the item selected in the drop-down list.

**12** Set the `InnerHtml` property of the `span` control to the date based on the current culture.



**13** Save the file and request it from the Web server.

**14** Click  to select a culture.

**15** Click the Submit button.

The message appears according to the region selected.

# LOCALIZE WITH THE PAGE CONTROL

You can use the Page control to specify a certain culture for individual pages in your ASP.NET application.

The Page control can be used for setting properties for a Web page. Culture is one of the properties that you can set for a page.

When using the Page control to localize, each culture that you support must have a separate Web page for that culture. This enables different team members to work on various parts of your Web application without affecting each other.

You can create a separate page for each culture. You must have a method of determining which culture to

use. You can allow the user to select a culture from a drop-down list. Based on this selection, you can use a case statement to redirect the user to the appropriate page. On these pages, you can use the @Page directive's Culture attribute to set the culture. When displaying the date on the page, you can use the CultureInfo.CurrentCulture to ensure the date is formatted properly.

You can have the user select a culture from a drop-down list of cultures and redirect them to an appropriate page based on the culture. Each page will be set up for different cultures and display the localized date.

## LOCALIZE WITH THE PAGE CONTROL

```

<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Select your culture below.

<FORM RUNAT="Server">
<P/>
<ASP:DROPDOWNLIST ID="dropdownlistCultures" RUNAT="Server">
  <ASP:LISTITEM VALUE="en-US">English</ASP:LISTITEM>
  <ASP:LISTITEM VALUE="es-ES">Spanish</ASP:LISTITEM>
  <ASP:LISTITEM VALUE="ja-JP">Japanese</ASP:LISTITEM>
</ASP:DROPDOWNLIST>

</FORM>

</FONT>
</BODY>

```

**1** Open GenericTemplate.aspx from the Code Templates directory.

**2** Add a heading to the page.

**3** Add a message to the user.

**4** Add a form control.

**5** Create a drop-down list that has several languages from which to choose.

```

<%@Page Language="C#" ResponseEncoding="utf-8" %>
<%@Import Namespace="System.Threading"%>
<%@Import Namespace="System.Globalization"%>
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimegoals.com</H3>
Select your culture below.

<FORM RUNAT="Server">
<P/>
<ASP:DROPDOWNLIST ID="dropdownlistCultures" RUNAT="Server">
  <ASP:LISTITEM VALUE="en-US">English</ASP:LISTITEM>
  <ASP:LISTITEM VALUE="es-ES">Spanish</ASP:LISTITEM>
  <ASP:LISTITEM VALUE="ja-JP">Japanese</ASP:LISTITEM>
</ASP:DROPDOWNLIST>
<ASP:BUTTON TEXT="Submit" OnClick="SubmitBtn_Click" RUNAT="Server"/>
<P/>
<SPAN ID="spanMessage" RUNAT="Server" />
</FORM>

```

**6** Add a submit button control.

**7** Add a span control on the form to display messages.

**8** Add the @Page directive to the page and set the ResponseEncoding attribute to UTF-8, as well as the Language attribute to C#.

**9** Add an alias to the System.Threading and System.Globalization namespaces.

## Apply It

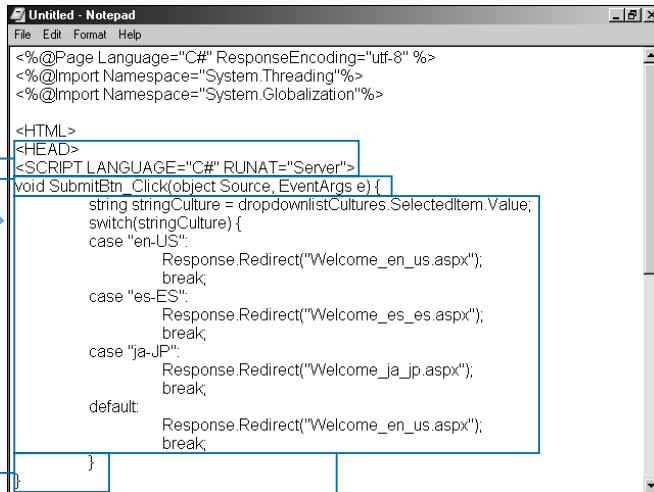
The `@Page` directive has a `Culture` attribute that you can set to the culture. Search MSDN for the `CultureInfo` class for more information on possible values for the `Culture` attribute.

### TYPE THIS:

```
<%@Page CULTURE="es" LANGUAGE="C#" %>
<%@Import Namespace="System.Globalization"%>
<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE="Verdana">
<H3></H3>
The local date is: <%=DateTime.Now.ToString("D",
CultureInfo.CurrentCulture) %>
</FONT>
</BODY>
</HTML>
```

### RESULT:

The local date is:  
domingo, 05 de agosto  
de 2001



```

<%@Page Language="C#" ResponseEncoding="utf-8" %>
<%@Import Namespace="System.Threading"%>
<%@Import Namespace="System.Globalization"%>

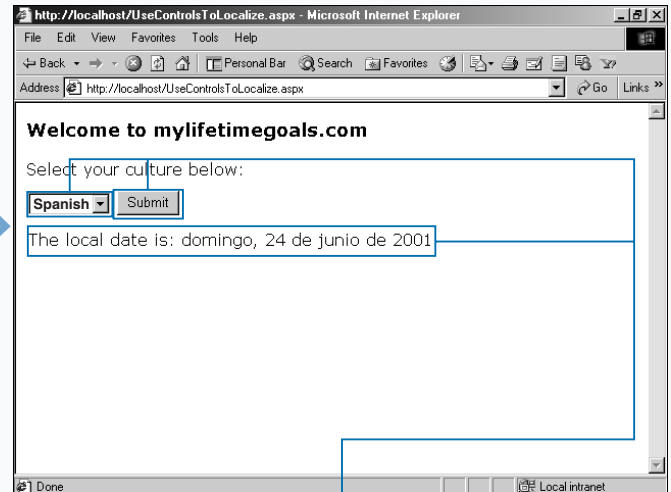
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
void SubmitBtn_Click(object Source, EventArgs e) {
    string stringCulture = dropdownListCultures.SelectedItem.Value;
    switch(stringCulture) {
        case "en-US":
            Response.Redirect("Welcome_en_us.aspx");
            break;
        case "es-ES":
            Response.Redirect("Welcome_es_es.aspx");
            break;
        case "ja-JP":
            Response.Redirect("Welcome_ja_jp.aspx");
            break;
        default:
            Response.Redirect("Welcome_en_us.aspx");
            break;
    }
}

```

**10** Add the `SubmitBtn_Click` function.

**11** Create a new string variable and read the value of the selected item in the drop-down list.

**12** Add a `switch` statement to redirect the user to the appropriate page.



**13** Save the file and request it from the Web server.

**14** Copy the `Welcomeen_us.aspx`, `Welcome_es_es.aspx`, and `Welcome_ja_jp.aspx` files from the CD-ROM to the current directory.

*Note: These files are used for the different cultures you can select.*

**15** Click  to select a culture.

**16** Click the Submit button.

The appropriate page for the culture appears with a message.

# CREATE AND USE RESOURCES

You can use *resource files* to store localization information for your ASP.NET Web application, which enables you to separate all of the information that is specific to a language or a locale from the application functions. This capability makes it easier to add more languages to the application.

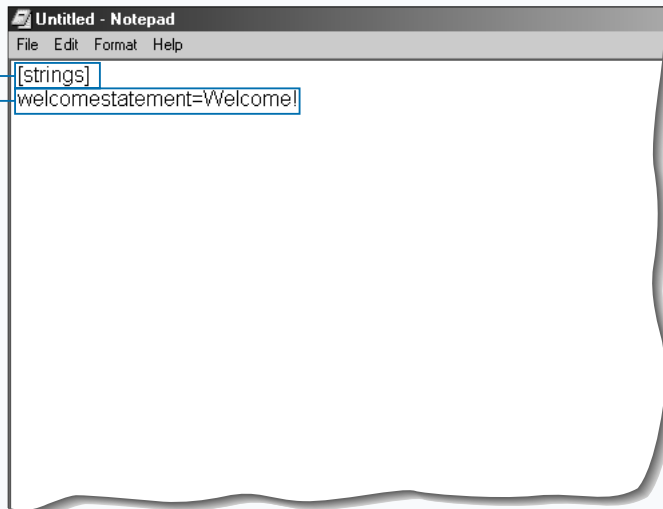
Resource files are composed of name/value pairs. Once you have created the resource file, you need to convert the file to a binary output by using the `resgen` command at the command line. To access the information in the resource file, you can use a class utility called a Resource Manager. You can load the

Resource Manager into an application variable by using the `Application_OnStart` event handler in the `Global.asax`.

You can save the resource files in a subdirectory to organize your files. If you want to do this, you can use the `Server.MapPath` function in the `Global.asax` to map to the subdirectory.

You can create a resource file and compile the resource file using the `resgen` command utility. You can then create a `Global.asax` file that will load the resources into a Resource Manager.

## CREATE AND USE RESOURCES

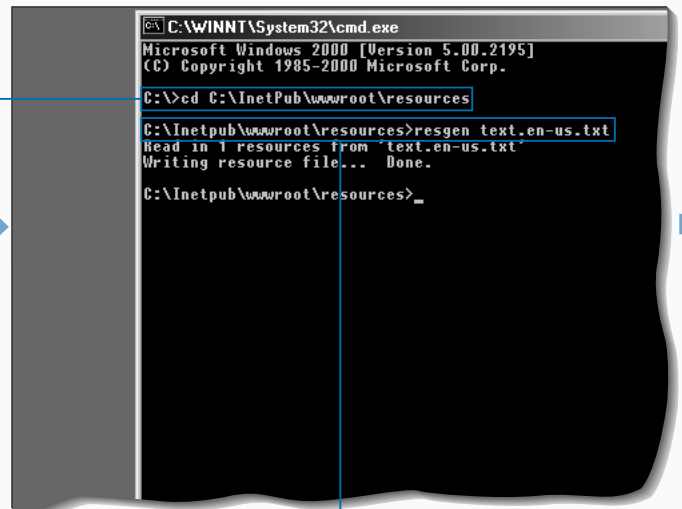


- 1 Open your text editor.
- 2 Type `[strings]`.
- 3 Add the `welcomestatement=Welcome!` name/value pair.
- 4 Save the file with a `.txt` extension.

- Repeat steps 2 to 4 for each culture.

*Note: The resource files are saved in a subdirectory called `resources`.*

*Note: This task works with three cultures (English, Spanish, and Japanese). You can expand to other cultures.*



- 5 Open the command prompt and go to the directory where your resource files are located.

- 6 Type the `resgen` command to create the resource file from the text file.

- Repeat step 6 for each resource file.

**Apply It**

You can try and read the user language from the Web browser. You can put the following function in your `Global.asax` file to set the culture for all requests. It attempts to read the user language if possible and sets the language to `en-us` for a default value.

**TYPE THIS:**

```
void Application_BeginRequest(Object sender, EventArgs args) {
    try {
        Thread.CurrentThread.CurrentCulture =
        new CultureInfo(Request.UserLanguages[0]);
    }
    catch(Exception) {
        Thread.CurrentThread.CurrentCulture =
        new CultureInfo("en-us");
    }
    Thread.CurrentThread.CurrentUICulture =
    Thread.CurrentThread.CurrentCulture;
}
```

**RESULT:**

The culture is set based on information that was sent from the user's Web browser.

```
Untitled - Notepad
File Edit Format Help
<%@Import Namespace="System.Globalization"%>
<%@Import Namespace="System.Resources"%>
<%@Import Namespace="System.Threading"%>
<%@Import Namespace="System.IO"%>

<SCRIPT RUNAT="Server" LANGUAGE="C#">
void Application_OnStart() {
    Application["applicationResourceManager"] =
    ResourceManager.CreateFileBasedResourceManager("text",
    Server.MapPath("resources")
    + Path.DirectorySeparatorChar,
    null);
}

</SCRIPT>
```

**7** Open your text editor to create a `Global.asax` file.

**8** Add aliases to `System.Globalization`, `System.Resources`, `System.Threading`, and `System.IO`.

```
Untitled - Notepad
File Edit Format Help
<%@Import Namespace="System.Globalization"%>
<%@Import Namespace="System.Resources"%>
<%@Import Namespace="System.Threading"%>
<%@Import Namespace="System.IO"%>

<SCRIPT RUNAT="Server" LANGUAGE="C#">
void Application_OnStart() {
    Application["applicationResourceManager"] =
    ResourceManager.CreateFileBasedResourceManager("text",
    Server.MapPath("resources")
    + Path.DirectorySeparatorChar,
    null);
}

</SCRIPT>
```

**9** Create the `Application_OnStart` event handler.

**10** Create a Resource Manager application variable and initialize it.

**11** Save the file as `Global.asax` to the Web site.

You can now use the Application variable to access the information in the resource files.

# USE RESOURCE MANAGER INFORMATION

You can create ASP.NET Web pages that use the information stored in the Resource Manager. The Resource Manager is a class utility that you can use to access the information stored in your resource files.

The culture for a user must be determined. One way you can determine this is to have the user select their culture from drop-down list. You need to make sure the response encoding is appropriate for the languages you are working with in your application. You can use UTF-8 to support multiple languages. You can add aliases to any namespaces for convenience.

You can use the `Page_Init` function for reading the Resource Manager variable out of the application variable. When you have access to the `ResourceManager` object, you can pull the appropriate string out of the Resource Manager according to the culture. Once you have the string, you can then update the `<span>` on the page to display the localized statement.

You can build the ASP.NET Web page that will be used to access the data in the resource file and display a statement based on the culture selected.

## USE RESOURCE MANAGER INFORMATION

```

Untitled - Notepad
File Edit Format Help
<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimagoals.com</H3>
Select your culture below.

<FORM RUNAT="Server">
<P>
<ASP.DROPDOWNLIST ID="dropdownlistCultures" RUNAT="Server">
  <ASP.LISTITEM VALUE="en-US">English</ASP.LISTITEM>
  <ASP.LISTITEM VALUE="es-ES">Spanish</ASP.LISTITEM>
  <ASP.LISTITEM VALUE="ja-JP">Japanese</ASP.LISTITEM>
</ASP.DROPDOWNLIST>
<ASP.BUTTON TEXT="Submit" OnClick="SubmitBtn_Click" RUNAT="Server"/>
<P>
<SPAN ID="spanMessage" RUNAT="Server" />
</FORM>

</FONT>
</BODY>

```

```

Untitled - Notepad
File Edit Format Help
<%@Page Language="C#" ResponseEncoding="utf-8" %>
<%@Import Namespace="System.Globalization"%>
<%@Import Namespace="System.Resources"%>
<%@Import Namespace="System.Threading"%>

<HTML>
<HEAD>

<SCRIPT LANGUAGE="C#" RUNAT="Server">
  ResourceManager resourcemanager>WelcomeStatement;
</SCRIPT>
</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3>Welcome to mylifetimagoals.com</H3>
Select your culture below.

<FORM RUNAT="Server">
<P>
<ASP.DROPDOWNLIST ID="dropdownlistCultures" RUNAT="Server">
  <ASP.LISTITEM VALUE="en-US">English</ASP.LISTITEM>
  <ASP.LISTITEM VALUE="es-ES">Spanish</ASP.LISTITEM>

```

- 1 Open the `GenericTemplate.aspx` template from the Code Templates directory.
- 2 Add a heading to the page.
- 3 Add a message to the user.
- 4 Add a form control.

- 5 Create a drop-down list that has several languages from which to choose.
- 6 Add a submit button control.
- 7 Add a span control on the form to display messages.

- 8 Add the `@Page` directive to the page and set the `ResponseEncoding` attribute to UTF-8, as well as the `Language` attribute to C#.

- 9 Add an alias to the `System.Globalization`, `System.Resources`, and `System.Threading` namespaces.
- 10 Add the `<script>` tags.
- 11 Create a new `ResourceManager` variable.



**Extra**

You can convert your resource files from their binary format into XML-formatted files. For example, you can use the `resgen` at the command line to convert the `text.en-us.resources` file into an XML file named `text.en-us.resx`.

**Example:**

```
resgen text.en-us.resources text.en-us.resx
```

You can convert from XML back to text as well. For example, you can use `resgen` at the command line to convert the `text.en-us.resx` file into an XML file named `newtext.en-us.txt`.

**Example:**

```
resgen text.en-us.resources newtext.en-us.resx
```

```

Untitled - Notepad
File Edit Format Help
<%@Page Language="C#" ResponseEncoding="utf-8" %>
<%@Import Namespace="System.Globalization"%>
<%@Import Namespace="System.Resources"%>
<%@Import Namespace="System.Threading"%>
<HTML>
<HEAD>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
ResourceManager resourceManager>WelcomeStatement;

void Page_Init(Object sender, EventArgs args) {
    resourceManager>WelcomeStatement = (ResourceManager)
Application["applicationResourceManager"];
}

void SubmitBtn_Click(object Source, EventArgs e) {
    string stringCulture = dropdownListCultures.SelectedItem.Value;
    Thread.CurrentThread.CurrentCulture = new CultureInfo(stringCulture);
    Thread.CurrentThread.CurrentUICulture = new CultureInfo(stringCulture);
    spanMessage.InnerHtml =
resourceManager>WelcomeStatement.GetString("welcomestatement");
}
</SCRIPT>
</HEAD>
<BODY>

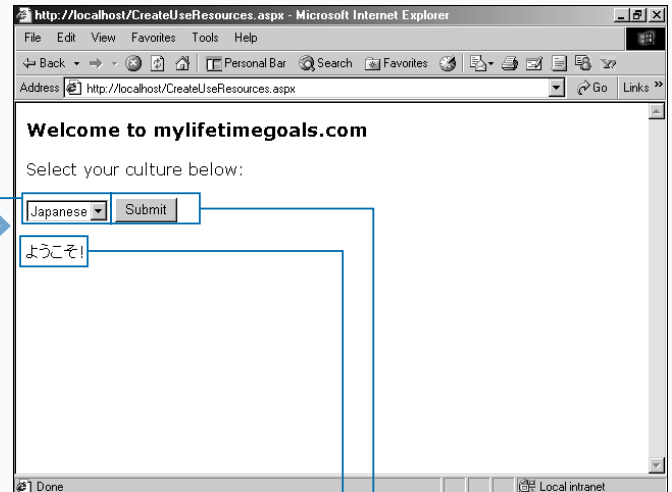
```

**12** Add the `Page_Init` event handler that reads the Resource Manager from the application variable.

**13** Create the `SubmitBtn_Click` function.

**14** Set the `CurrentCulture` and `CurrentUICulture` based on the selection in the drop-down list.

**15** Set the message using the Resource Manager.



**16** Save the file and request it from the Web server.

**17** Click  to select a culture.

**18** Click the Submit button.

The message is localized according to the culture selected.

# WORK WITH MULTIPLE SERVER-SIDE LANGUAGES

You can implement ASP.NET applications that use multiple server-side languages. The .NET platform supports several languages and is architected to support any mainstream language that can supply a compiler that will generate IL (Intermediate Language) that is compatible with the CLR (Common Language Runtime). Also, if the language is to interoperate with other .NET languages, it must be CLS (Common Language Specification) compliant.

The .NET platform does support multiple languages, and ASP.NET applications can use these CLR compliant languages, but the Web pages only support

one server-side language per page. The one language per Web page applies only to code processed on the server. You can still have multiple client-side languages. ASP.NET treats the client-side language just like normal HTML markup and lets the user's browser interpret the script. Client-side language support is handled by the browser, just as with ASP 3.0 applications.

To indicate what language you want to run on a page for server-side code processing, you can set it with the `Language` attribute on the `@Page` directive. If the language is not specified, ASP.NET assumes that the language is VB.

## WORK WITH MULTIPLE SERVER-SIDE LANGUAGES

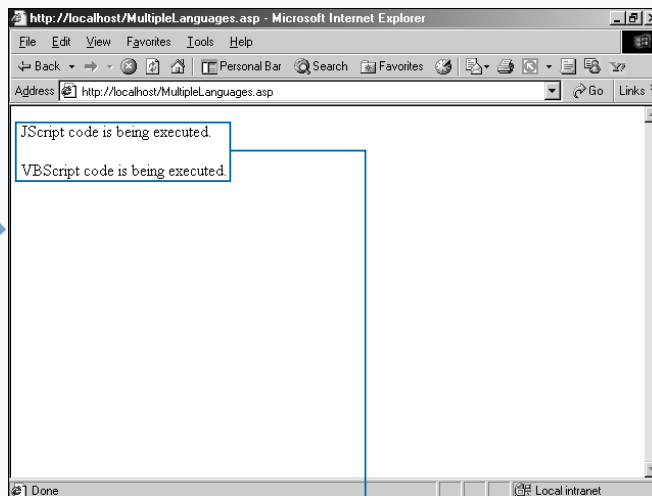
```

<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3></H3>
<FORM RUNAT="Server">
<P/>
<SCRIPT LANGUAGE="JScript" RUNAT="Server">
    Response.Write("JScript code is being executed.");
</SCRIPT>
<P/>
<SCRIPT LANGUAGE="VBScript" RUNAT="Server">
    Response.Write "VBScript code is being executed."
</SCRIPT>
</FORM>

```

**1** Open `MultipleLanguages.aspx` from the Code Templates directory.

The page displays two server-side script blocks using two different languages.



**2** Save the file and request it from the Web server.

The JScript code and the VBScript code execute.

**Extra**

ASP.NET Web pages are compiled on the Web server. This is why there is support for only one language per page. This is different than traditional ASP pages, which are normally interpreted each time they are requested.

If the language is not specified in the @Page directive and you only have server-side code in a `<script>` tag, then you could set the `Language` attribute for the `script` element.

The .NET Framework provides the Common Language Specification (CLS) to ensure that .NET compliant language can interoperate. CLS describes a fundamental set of language features and defines rules for how those features are used. CLS-compliant languages enable you to do such things as inherit classes from other CLS-compliant classes and pass data types without having to do any special preparation (like buffering a string).

When you know how to program in a language that is CLS compliant, you can leverage this knowledge by creating other applications that run on the CLR (like a Windows Forms application, a Web Service, a Mobile application, or a distributed application).

```

<HTML>
<HEAD>
</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3></H3>
<FORM RUNAT="Server">
<P/>
<SCRIPT LANGUAGE="C#" RUNAT="Server">
    Response.Write("C# code is being executed.");
</SCRIPT>
<P/>
<SCRIPT LANGUAGE="VB" RUNAT="Server">
    Response.Write "VB code is being executed."
</SCRIPT>
</FORM>

```

**3** Open `MultipleLanguages.aspx` from the Code Templates directory.

The page displays two server-side script blocks using two different languages.

**Server Error in '/' Application**

**Parser Error Message:** Cannot use language 'VB' because another language has been specified earlier in this page

**Source Error:**

```

Line 13: </SCRIPT>
Line 14: <P/>
Line 15: <SCRIPT LANGUAGE="VB" RUNAT="Server">
Line 16:     Response.Write "VB code is being executed."
Line 17: </SCRIPT>

```

**Source File:** c:\inetpub\wwwroot\MultipleLanguages.aspx **Line:** 15

**4** Save the file and request it from the Web server.

An error message appears because more than two languages cannot be used on the same page.

# WORK WITH SCRIPT BLOCKS

You can embed server-side code in ASP.NET and ASP applications with `<script>` blocks. This feature has not changed from ASP applications to ASP.NET applications. Server-side `<script>` blocks were available and configured the same way. What is new is the concept of using code-behind pages in ASP.NET pages. See page 192 for a cleaner way to implement server-side code.

What you need to be aware of with `<script>` blocks is that they are necessary for procedures that are placed inside of the ASP.NET page. Before, with ASP

pages, you could put procedures within code delimiters (`<%` and `%>`) without issues. In ASP.NET, however, this is not allowed.

To include a server-side `<script>` block, you need to (at a minimum) put in the `runat="Server"` attribute. You also should be explicit with the language you are implementing inside of the `<script>` block. The default is VB. So, if you desire to use C#, you will need to put the `Language="C#"` attribute inside the `script` element.

## WORK WITH SCRIPT BLOCKS

```

<%@ Language="VBScript" %>

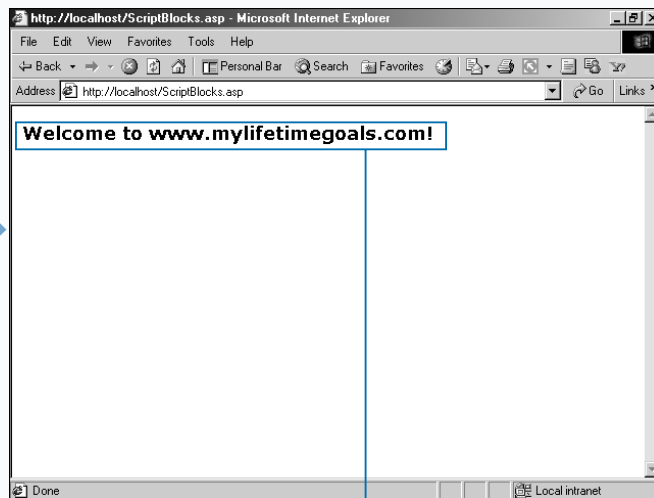
<HTML>
<HEAD>
<%
  Sub SayWelcomeStatement()
    Response.Write "Welcome to www.mylifetimegoals.com!"
  End Sub
%>

</HEAD>
<BODY>
<FONT FACE ="Verdana">
<H3><%SayWelcomeStatement%></H3>
</FONT>

```

**1** Open `ScriptBlocks.aspx` from the Code Templates directory.

The page displays the `SayWelcomeStatement` that outputs the header for the page.



**2** Save the file and request `ScriptBlocks.aspx` from the Web server.

The ASP code executes `ScriptBlocks.aspx` without issues.

**Extra**

You can put scripts into files that are external to your ASP.NET page. You can do this with the `<script>` element by specifying an external script file using the `src` attribute. When you define the `src` attribute, all content between the opening and closing tags of the `<script>` element is ignored. Because this is the case, it is best to define the `<script>` element as an empty tag. For example, `<script runat="server" src="scrStandard.cs" />`.

With ASP.NET, you must declare global variables within `<script runat=server>` blocks and not between code delimiters (`<%` and `%>`).

```

<%@ Language="VB" %>

<HTML>
<HEAD>
<%
  Sub SayWelcomeStatement()
    Response.Write "Welcome to www.mylifetimegoals.com!"
  End Sub
%>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<H3><%SayWelcomeStatement%></H3>

</FONT>

```

**3** Open `ScriptBlocks.aspx` from the Code Templates directory.

The page displays the `SayWelcomeStatement` that outputs the header for the page.

**Compilation Error**

**Description:** An error occurred during the compilation of a resource required to service this request. Please review the specific error details below and modify your source code appropriately.

**Compiler Error Message:** BC30289: This statement cannot appear within a method body. The compiler will assume an intent to terminate the method body.

**Source Error:**

```

Line 4: <HEAD>
Line 5: <%
Line 6:   Sub SayWelcomeStatement()
Line 7:     Response.Write "Welcome to www.mylifetimegoals.com!"
Line 8:   End Sub

```

**4** Save the file and request it from the Web server.

An error message appears because parentheses were not used when calling the subprocedure.

# USING RENDER FUNCTIONS

**R**ender functions that were in your ASP applications can be replaced with global functions in ASP.NET. Traditional render functions that were available in ASP applications are no longer available in ASP.NET applications. Render functions were functions that embedded HTML inside of the function. This HTML was not written with the `Response.Write`; it was embedded inside of the function. This was done by starting the subroutine (`<% Sub RenderHeader() %>`) with an open code delimiter and ending with a close code delimiter. Then the HTML would be written as though it were outside of the function (`<H1>Welcome to www.mylifetimegoals.com</H1>`). Finally, the subroutine would be closed with code

delimiters (`<% End Sub %>`). This render function could be called conditionally after the subroutine or function definition (`<% Call RenderHeader %>`).

This ASP trick is no longer available in ASP.NET. There are two issues with render functions that violate ASP.NET syntax rules. First, HTML can not be embedded in a function. To resolve this issue, use `Response.Write`. Second, procedures cannot reside inside of code delimiters (`<%` and `%>`). This is resolved by putting the function or subroutine in the server-side `<script>` blocks.

## USING RENDER FUNCTIONS

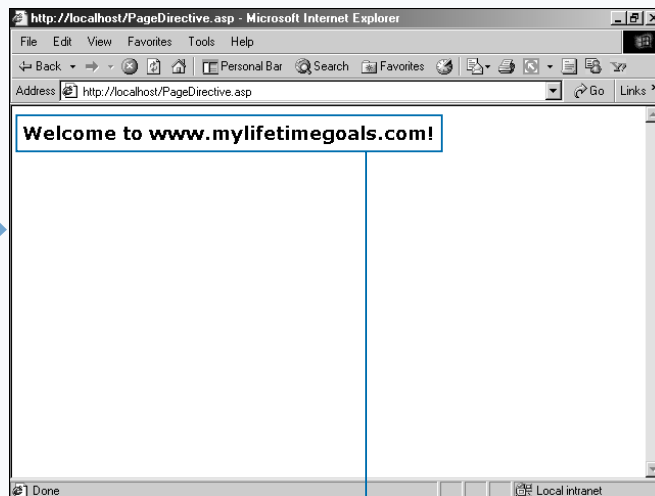
```

Untitled - Notepad
File Edit Format Help
<%@ Language="VBScript" %>

<HTML>
<HEAD>
<% Sub SayWelcomeStatement() %>
  <H3>Welcome to www.mylifetimegoals.com!</H3>
<% End Sub %>

</HEAD>
<BODY>
<FONT FACE ="Verdana">
<% SayWelcomeStatement %>
</FONT>
</BODY>
</HTML>

```



**1** Open `RenderFunctions.aspx` from the Code Templates directory.

The page displays the `SayWelcomeStatement` that outputs the header for the page.

**2** Save the file and request it from the Web server.

The ASP code is able to use the render function.

**Extra**

Rendering functions were commonplace in ASP applications. Using rendering functions to build tables from recordsets was very convenient, but it created very messy code that was hard to troubleshoot. Because rendering functions are not allowed in ASP.NET, you can either put the functions in a script block using the `Response.Write ("html goes in here")` method, or you can use the data binding capabilities that come with the ASP.NET framework. Refer to Chapter 6 for data binding to server-side controls. This gives you the ability to populate a table with much less code that is easier to follow.

```

<%@ Page LANGUAGE="VB" %>

<HTML>
<HEAD>
<% Sub SayWelcomeStatement() %>
  <H3>Welcome to www.mylifetimegoals.com!</H3>
<% End Sub %>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

<% SayWelcomeStatement() %>

</FONT>
</BODY>
</HTML>

```

**3** Open `RenderFunctions1.aspx` from the Code Templates directory.

The page displays the `SayWelcomeStatement` that outputs the header for the page.

**Compilation Error**

**Description:** An error occurred during the compilation of a resource required to service this request. Please review the specific error details below and modify your source code appropriately.

**Compiler Error Message:** BC30289: This statement cannot appear within a method body. The compiler will assume an intent to terminate the method body.

**Source Error:**

```

Line 3: <HTML>
Line 4: <HEAD>
Line 5: <% Sub SayWelcomeStatement() %>
Line 6:   <H3>Welcome to www.mylifetimegoals.com!</H3>
Line 7: <% End Sub %>

```

**4** Save the file and request it from the Web server.

An error message appears because a render function was used.

# USING PAGE DIRECTIVES

The standard set of processing directives for an ASP.NET page has changed dramatically from what was available with ASP. When migrating from ASP to ASP.NET, you need to update your pages to use the `@Page` directive to let ASP.NET know certain processing requirements for an ASP.NET Web page. For example, in ASP pages, if you wanted to specify the server-side language, you used the `@Language` directive. Now, with ASP.NET Web pages, you must use the `Language` attribute that is available via the `@Page` directive.

The `@Page` directive contains many of the directives that were available for ASP Applications. `LCID` and

`CodePage` attributes on the `@Page` directive for ASP.NET Web pages are all directives that must be placed on the first line of a page within the same delimiting block, for example, ASP.NET `<%@Page Language="VB" CodePage="932"%>` versus ASP `<%@ LANGUAGE="VBSCRIPT" CODEPAGE="932"%>`.

With ASP.NET Web pages, you can have as many lines of directives as you need. Standard practice is to put directives at the top of the page, but you are able to put them anywhere in your ASP.NET Web page.

## USING PAGE DIRECTIVES

```

<%@ LANGUAGE = "VBScript" ENABLESESSIONSTATE = "True" %>

<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE ="Verdana">

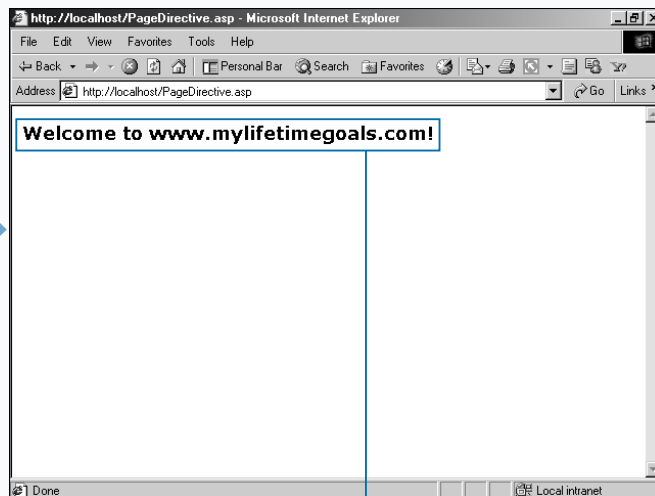
<H3>Welcome to www.mylifetimegoals.com!</H3>

</FONT>
</BODY>
</HTML>

```

**1** Open `PageDirective.aspx` from the Code Templates directory.

The page does not use the `@Page` directive. It uses the `@Language` directive without the `@Page`.



**2** Save the file and request it from the Web server.

The ASP code executes displaying the header that is contained in a subroutine.



**Extra**

When migrating ASP applications to ASP.NET applications, you have other directives besides the `@Page` directive. For example, you can use the `@OutputCache` directive to control how a page is cached on the server. With page caching on the server, results of a processed `.aspx` page are held in memory at the server. The next time the page is requested, the cached page can be sent as the response instead of regenerating the page. This can give you excellent performance gains, especially when the page generation involves calls to other machines (like database servers). You can control the location of where the page is cached via the `Location` attribute. The `Location` attribute has the following options:

OPTION	LOCATION
Any	Client, Downstream, or Server
Client	Browser client where the request originated
Downstream	A server downstream from the Web server that processed the request
None	N/A
Server	Web server where the request was processed.

```

<%@ PAGE LANGUAGE = "VBScript" ENABLESESSIONSTATE = "True" %>

<HTML>
<HEAD>

</HEAD>
<BODY>
<FONT FACE = "Verdana">

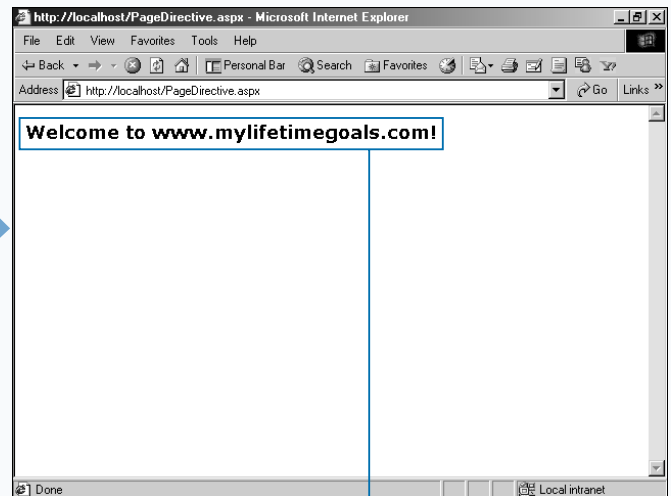
<H3>Welcome to www.mylifetimegoals.com!</H3>

</FONT>
</BODY>
</HTML>

```

**3** Open `PageDirective.aspx` from the Code Templates directory.

The page uses the `@Page` directive to set the server-side language.



**4** Save the file and request it from the Web server.

The ASP code executes using the `@Page` directive to determine the server-side language.

# MIGRATE VBSCRIPT TO VB.NET SYNTAX

For your server-side code, you can migrate VBScript code to VB.NET in your ASP.NET applications. VBScript is the most common language used for ASP Applications. If you decide to migrate an ASP application that used VBScript for server-side code, you can make some minor modifications to VBScript to turn it into VB code that will run in the Common Language Runtime (CLR).

If you migrate an existing ASP application to an ASP.NET application, you will have to make some decisions on how far you want to take your conversion process. At a high level, you can take one of three paths depending on the number of changes

you want to make. The first path is to rewrite your entire application, treating you ASP application as a prototype. This has the most cost initially, but it may be the cheapest option in the long run. The second option is to just convert the ASP pages (\*.asp) to ASP.NET pages (\*.aspx) — making the minimal amount of changes required to give the file an .aspx extension. The third option, initially the cheapest cost of migration, is to leave the file as an ASP page, keeping the extension as .asp. A migration project can involve one or more of these paths. Most likely, you will treat this on a page-by-page basis.

## MIGRATE VBSCRIPT TO VB.NET SYNTAX

```

<%@ LANGUAGE = VBScript %>
<% Option Explicit %>

<HTML>
<HEAD>
<TITLE>Simple ADO Query</TITLE>
</HEAD>

<BODY BGCOLOR="White" topmargin="10" leftmargin="10">

<!-- Display Header -->

<font size="4" face="Arial, Helvetica">
<b>Simple ADO Query with ASP</b></font><br>

<hr size="1" color="#000000">

Contacts within the Authors Database:<br><br>

<%
    Dim oConn
    Dim oRs
    Dim filePath
    Dim Index
  
```

Simple ADO Query with ASP

Contacts within the Authors Database:

73	Scott Guthrie	1975
114	Shammas, Namir Clement	1954
244	Vaughn, William	1947
611	Bard, Dick	1941
1410	Davis, Steve	1953
2779	Vaughn, William R.	1947
3193	Wraye, Toby	1950
3461	Simpson, Alan	1953
3495	Jones, Edward	1952
3915	Grommes, Bob	1957
3924	Pfaffenberger, Bryan	1949

**1** Open SimpleQuery\_VBScript.asp from the Code Templates directory.

**2** Scroll down the file to view the code.

**3** Save the file and request SimpleQuery\_VBScript.asp from the Web server.

**4** Copy the file from the CD-ROM to the working directory.

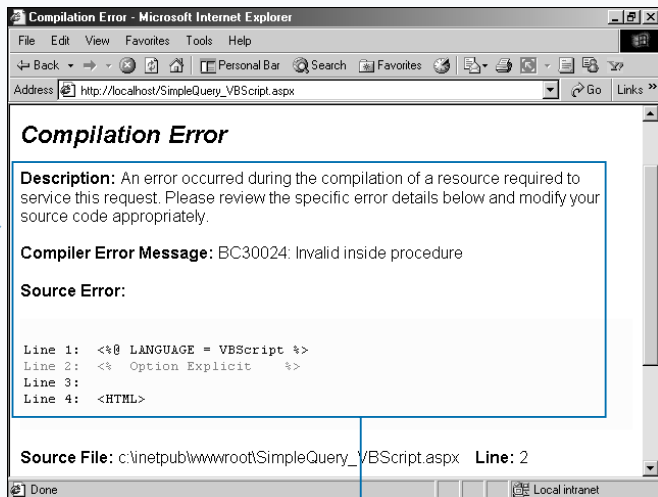
The ASP code executes without issues and displays data from the database of the author.

**Extra**

Microsoft has made VB a first class language, keeping the following in mind:

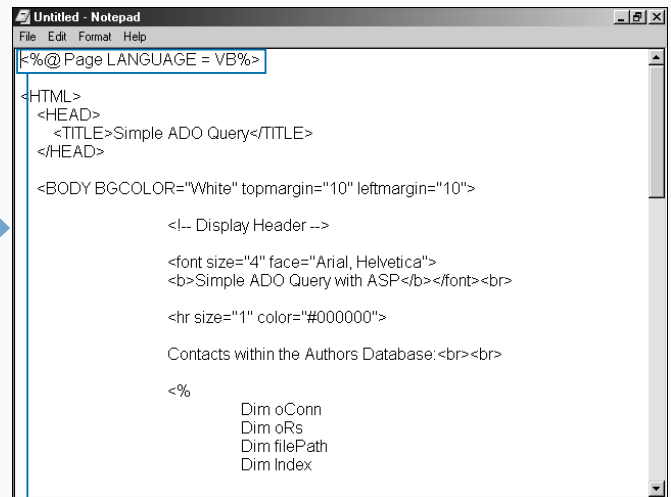
- Making the language more consistent: bringing together features of the language with similar purpose.
- Simplifying the language: redesigning those features which made Visual Basic anything less than "basic."
- Improving readability and maintainability: redesigning features that hide too many important details from the programmer.
- Improving robustness: enforcing better practices, such as type-safe programming.

In VB.NET, the `On Error Resume Next` and `On Error Goto` error handling is still available. Even though this is still available, you should take advantage of structured error handling that is available for all CLS (Common Language Specification) compatible languages. This uses the `Try`, `Catch`, and `Finally` keywords.



**5** Save the file with an `.aspx` extension and then request it from the Web server.

An error message appears because the `Option Explicit` is not allowed.



**6** Change the language in the source file from `VBScript` to `VB`, add the keyword `Page` to the directive at the top of the file, and remove the `Option Explicit` code.

**7** Request `SimpleQuery_VBScript.aspx` from the Web server.

An error message appears because the `Let` and `Set` statements are no longer supported.

CONTINUED

# MIGRATE VBSCRIPT TO VB.NET SYNTAX

Not all ASP pages need to be changed when converting to an ASP.NET application. However, for the ASP pages that you will convert to ASP.NET, you have to first change the extension from `asp` to `aspx`. After the extension is changed, you must change the processing directives on the ASP page. Most of the directives on an ASP page will migrate to the `@Page` directive, for example, `Language=VBScript` will change to `@Page Language=VB`. See page 280 for further details on the `@Page` directive).

A big change from VBScript to VB.NET is the removal of the variant type. In VBScript, you did not have the

ability to strongly type variables, but you would use `Option Explicit` to avoid using variables that were not declared. If you mistyped a variable name and did not use `Option Explicit`, you could generate some interesting bugs. `Option Explicit` is no longer needed because it is the default for ASP.NET pages.

If you decide to interoperate with COM objects, you need to be aware that ASP.NET's threading model is the Multiple Threaded Apartment (MTA). This means that a standard ASP.NET page will not be able to use a normal COM object created by VB6, which is Single Threaded Apartment (STA). To correct this you can set `AspCompat="true"` for the `@Page` directive.

## MIGRATE VBSCRIPT TO VB.NET SYNTAX (CONTINUED)

```

' Map authors database to physical path
filePath = Server.MapPath("authors.mdb")

' Create ADO Connection Component to connect
' with sample database

oConn = Server.CreateObject("ADODB.Connection")
oConn.Open "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=" & filePath

' Execute a SQL query and store the results
' within recordset
oRs = oConn.Execute("SELECT * From authors")

%>

<TABLE border = 1>
<%
    Do while (Not oRs.eof) %>
  
```

```

' Map authors database to physical path
filePath = Server.MapPath("authors.mdb")

' Create ADO Connection Component to connect
' with sample database

oConn = Server.CreateObject("ADODB.Connection")
oConn.Open "Provider=(Microsoft.Jet.OLEDB.4.0;Data
Source=" & filePath)

' Execute a SQL query and store the results
' within recordset
oRs = oConn.Execute("SELECT * From authors")

%>

<TABLE border = 1>
<%
    Do while (Not oRs.eof) %>
  
```

**8** Remove the two `Set` statements from the source file.

**9** Save the file and request `SimpleQuery_VBScript.aspx` from the Web server.

An error message appears because VB requires the argument list to be enclosed in parentheses.

**10** Place parentheses around the statement `"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & filePath`.

**11** Save the file and request `SimpleQuery_VBScript.aspx` from the Web server.

An error message appears because you cannot create apartment threaded COM components without the `aspcompat=true` statement in the `@Page` directive.

**Extra**

If you are not explicit in how parameters are passed, you could have issues with your code. In VB6, parameters were, by default, passed by reference (ByRef). This has changed to by value (ByVal) in VB.NET. If your code relied on ByRef parameters, then you could introduced bugs with this change.

VB.NET no longer provides support for default properties. To account for this, you need to explicitly call the property from the object.

**Example:**

```
Response.Write oRS("FirstName")
```

'Changes to:

```
Response.Write (oRS("FirstName").Value)
```

All of your server-side code method calls will require parentheses.

**Example:**

```
Response.Write "An error has occurred."
```

'Changes to:

```
Response.Write("An error has occurred.")
```

```

Untitled - Notepad
File Edit Format Help
<%@ Page LANGUAGE = VB ASPCOMPAT=true %>

<HTML>
<HEAD>
<TITLE>Simple ADO Query</TITLE>
</HEAD>

<BODY BGCOLOR="White" topmargin="10" leftmargin="10">

<!-- Display Header -->

<font size="4" face="Arial, Helvetica">
<b>Simple ADO Query with ASP</b></font><br>

<hr size="1" color="#000000">

Contacts within the Authors Database:<br><br>

<%
    Dim oConn
    Dim oRs
    Dim filePath
    Dim Index
  
```

**12** Add `aspcompat=true` to the `@Page` directive at the top of the page.

**13** Save the file and request `SimpleQuery_VBScript.aspx` from the Web server.

■ The page renders without issues but does not display the data because the default properties are not supported in VB.

```

Untitled - Notepad
File Edit Format Help

        oRs = oConn.Execute("SELECT * From authors")
    %>

    <TABLE border = 1>
    <%
        Do while (Not oRs.eof) %>
            <tr>
                <% For Index=0 to
                    (oRs.fields.count-1) %>
                    <TD VAlign=top><%=
                    oRs(Index).Value %></TD>
                <% Next %>
            </tr>
            <% oRs.MoveNext
        %>
    Loop
    </TABLE>
  
```

**14** Add `.Value` to the end of the `oRs (Index)` statement.

**15** Save the file and request `SimpleQuery_VBScript.aspx` from the Web server.

■ The page displays properly as an ASP.NET page.

# MIGRATE JSCRIPT TO JSCRIPT.NET SYNTAX

JScript.NET code implementation is much closer to the JScript implementation than VB.NET is to VBScript. Because JScript was more of an Object-Oriented language than VBScript, JScript will resemble to the C# syntax that is used in this book than the VB syntax. The big difference that you will notice is in the variable declaration.

When converting an ASP page (\*.asp) to an ASP.NET page (\*.aspx) that has JScript in the server-side code, the first thing you will do is change the file extension to .aspx. If you are going to access STA (Single Threaded Apartment) COM components,

then you need to set `ASPCOMPAT=true` for the `@Page` directive.

As with VB.NET, you can not access default properties on COM objects. If default properties are used, you have to find the explicit property that retrieves the default property and program using that explicit property. For example, if you have a label on a form, you can not access the `Text` property by the default property of the label. To access the `Text` property, you have to give an explicit reference to that property's data type — JScript.NET - `var s : String = lblFirstName.Text`; versus JScript - `var s = lblFirstName`; Note that the `String` is explicitly declared when initializing the variable.

## MIGRATE JSCRIPT TO JSCRIPT.NET SYNTAX

```

Untitled - Notepad
File Edit Format Help
<%@LANGUAGE = JScript ASPCOMPAT =true %>

<HTML>
<HEAD>
<TITLE>Simple ADO Query</TITLE>
</HEAD>

<BODY BGCOLOR="White" topmargin="10" leftmargin="10">

<!-- Display Header -->

<font size="4" face="Arial, Helvetica">
<b>Simple ADO Query with ASP</b></font><br>

<hr size="1" color="#000000">

Contacts within the Authors Database:<br><br>

<%
var oConn;
var oRs;
var filePath;
  
```

```

Untitled - Notepad
File Edit Format Help

var filePath;
// Map authors database to physical path
filePath = Server.MapPath("authors.mdb");
// Create ADO Connection Component to connect with
// sample database
oConn = Server.CreateObject("ADODB.Connection");
oConn.Open("Provider=Microsoft.Jet.OLEDB.4.0;Data
Source="+filePath);
// Execute a SQL query and store the results within
// recordset
oRs = oConn.Execute("SELECT * From authors");
%>

<TABLE border = 1>
<%
while (!oRs.eof) { %>
  
```

**1** Open `SimpleQuery_JScript.aspx` from the Code Templates directory.

The `ASPCOMPAT=true` attribute is set in the `@Page` directive.

**2** Scroll down the file.

The `filePath` statement appears in parentheses, so you do not have to change it.

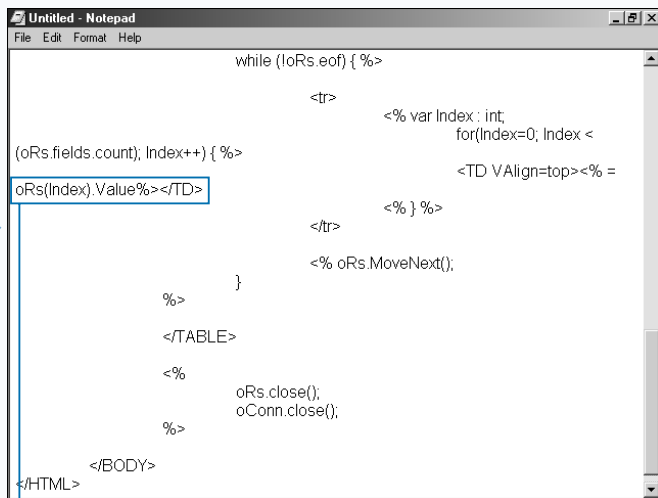
The `set` statement is not required in JScript, so there are no changes with initializing the object variable.

**3** Scroll down the file.

**Extra**

JScript.NET is Microsoft's implementation of the ECMA 262 language. Improvements in JScript.NET — which is being developed in conjunction with ECMAScript Edition 4 — include true compiled code, typed and typeless variables, classes (with inheritance, function overloading, property accessors, and more), packages, cross-language support, and access to the .NET Framework. JScript.NET is a true object-oriented scripting language. Even though JScript.NET can now use classes, types, and other “industrial strength” language features for writing robust applications, it still keeps its “scripting” feel.

ECMAScript has strong roots with Netscape. Having JScript is beneficial for those who are inline with the standards body that defines ECMAScript. If you are looking for features and support, you will find more of this with the C# and VB languages. There are more books and sample code for these languages.



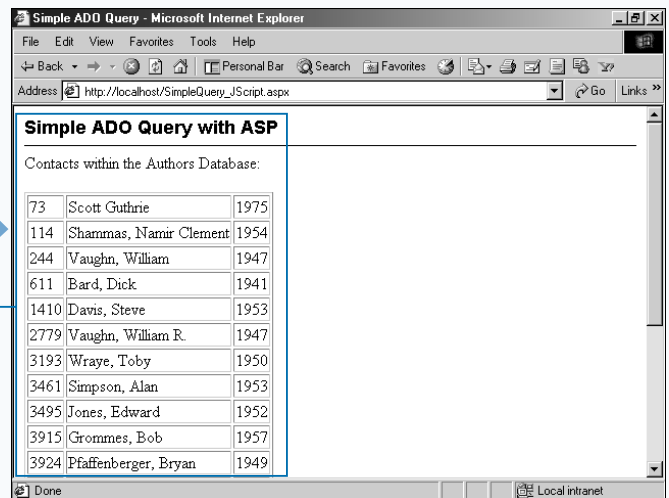
```

while (!oRs.eof) { %>
    <tr>
        <% var Index : int;
           for(Index=0; Index <
              <TD VAlign=top><% =
oRs(Index).Value%></TD>
        <% } %>
    </tr>
    <% oRs.MoveNext();
    }
%>
</TABLE>
<%
oRs.close();
oConn.close();
%>
</BODY>
</HTML>

```

■ The statement `oRs (Index)` was changed to `oRs (Index) .Value`.

■ Request the `SimpleQuery_JScript.aspx` from the Web server.



Simple ADO Query with ASP

Contacts within the Authors Database:

73	Scott Guthrie	1975
114	Shammas, Namir Clement	1954
244	Vaughn, William	1947
611	Bard, Dick	1941
1410	Davis, Steve	1953
2779	Vaughn, William R.	1947
3193	Wraye, Toby	1950
3461	Simpson, Alan	1953
3495	Jones, Edward	1952
3915	Grommes, Bob	1957
3924	Pfaffenberger, Bryan	1949

■ The page appears properly as an ASP.NET page.

# HTML SERVER CONTROLS

## BASIC HTML CONTROLS

NAMESPACE:	SYSTEM.WEB.UI.HTMLCONTROLS:
HtmlAnchor	working with the HTML <a> tag on the server.
HtmlButton	working with the HTML <button> tag on the server.
HtmlForm	working with the HTML <form> tag on the server.
HtmlGenericControl	working with tags not represented by .NET classes (eg. <span> and <div> tags).
HtmlImage	working with the HTML <img> tag on the server.
HtmlInputButton	working with the HTML <input type= button>, <input type= submit>, and <input type= reset> tags on the server.
HtmlInputCheckBox	working with the HTML <input type= checkbox> tag on the server.
HtmlInputFile	working with the HTML <input type= file> tag on the server.
HtmlInputHidden	working with the HTML <input type= hidden> tag on the server.
HtmlInputImage	working with the HTML <input type= image> tag on the server.
HtmlInputRadioButton	working with the HTML <input type= radio> tag on the server.
HtmlInputText	working with the HTML <input type= text> and <input type= password> tags on the server.
HtmlSelect	working with the HTML <select > tag on the server.
HtmlTable	working with the HTML <table> tag on the server.
HtmlTableCell	working with the HTML <td> and <th> tags on the server.
HtmlTableCellCollection	working with a collection of HTML <td> and <th> tags on the server.
HtmlTableRow	working with the HTML <tr> tag on the server.
HtmlTableRowCollection	working with a collection of HTML <tr> tags on the server.
HtmlTextArea	working with the HTML <textarea> tag on the server.



## WEB SERVER CONTROLS

NAMESPACE:	SYSTEM.WEB.UI.WEBCONTROLS:
AdRotator	displaying an advertisement banner.
Button	posting data back to the server.
Calendar	displaying a one-month calendar.
CheckBox	creating a check box.
CheckBoxList	creating a multi-selection check box group.
DataGrid	creating a multi-column data bound grid.
DataList	creating a data bound list.
DropDownList	creating a drop-down list that contains a single selection.
HyperLink	creating a link for navigating to pages.
Image	creating a Web-compatible image.
ImageButton	creating an image to handle user click events.
Label	creating and manipulating static text.
LinkButton	creating hyperlink-style buttons that post back to the same page on which they originated.
ListBox	displaying a single-selection or multi-selection list box.
Panel	providing a container for other controls.
RadioButton	creating a radio button.
RadioButtonList	creating a radio button group.
Repeater	creating a data-bound list that renders a row for every row in the specified data source.
Table Web	creating a table and manipulating it programmatically.
TableCell	creating a table cell and manipulating it programmatically.
TableRow	creating a table row and manipulating it programmatically.
TextBox	creating single and multi-line text boxes.

## VALIDATION SERVER CONTROLS

NAMESPACE:	SYSTEM.WEB.UI.WEBCONTROLS:
CompareValidator	comparing a user's entry against a constant value, against a property value of another control, or against a database value using a comparison operator (less than, equal, greater than, and so on).
CustomValidator	creating custom server and client validation code.
RangeValidator	checking that a user's entry is between specified lower and upper boundaries.
RegularExpressionValidator	checking that the entry matches a pattern defined by a regular expression.
RequiredFieldValidator	checking that the user does not skip an entry.
ValidationSummary	displaying a summary of all validation errors for all of the validation controls on a page.

## GLOBAL.ASAX SYNTAX

```
<script language="VB" runat=server>
  Sub Application_OnStart()
    ' Application startup code goes here...
  End Sub
  Sub Session_OnStart()
    ' Session startup code goes here...
  End Sub
  Sub Session_OnEnd()
    ' Session cleanup code goes here...
  End Sub
  Sub Application_OnEnd()
    ' Application cleanup code goes here...
  End Sub
</script>
```

## WEB.CONFIG SYNTAX

The following is an example of a template for the web.config file.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

  <system.web>

    <!-- DYNAMIC DEBUG COMPILATION
      Set debugmode enable="true" to enable ASPX debugging. Otherwise, setting this value to
      false will improve runtime performance of this application.
    -->
    <compilation
      defaultlanguage="c#"
      debug="true"
```

## WEB.CONFIG SYNTAX (CONTINUED)

```

/>
<!-- CUSTOM ERROR MESSAGES
Set mode enable="on" or "remoteonly" to enable custom error messages, "off" to disable. Add
<error> tags for each of the errors you want to handle.
-->
<customErrors
mode="Off"
/>

<!-- AUTHENTICATION
This section sets the authentication policies of the application. Possible modes are "Windows", "Cookie",
"Passport" and "None"
-->
<authentication mode="Forms">
    <forms name=".ASPXUSERDEMO" loginUrl="login.aspx" protection="all" timeout="60" />
</authentication>
<authorization>
    <deny users="?" />
</authorization>

<!-- 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="false"
requestLimit="0"
pageOutput="false"
traceMode="SortByTime"
/>

<!-- SESSION STATE SETTINGS
By default ASP+ uses cookies to identify which requests belong to a particular session.
If cookies are not available, a session can be tracked by adding a session identifier to the URL.
To disable cookies, set sessionState cookieless="true".
-->
<sessionState
mode="inproc"
stateConnectionString="tcpip=127.0.0.1:42424"
sqlConnectionString="data source=127.0.0.1;user id=sa;password="
cookieless="false"
timeout="20"
/>

<!-- GLOBALIZATION
This section sets the globalization settings of the application.
-->
<globalization
requestEncoding="utf-8"
responseEncoding="utf-8"
/>

</system.web>

</configuration>

```

# BASICS EXAMPLES

## DECLARING VARIABLES

### Visual Basic

```
Dim x As Integer
Public x As Integer = 10
```

### C#

```
int x;
int x = 10;
```

### JScript

```
var x : int;
var x : int = 10;
```

## COMMENTS

### Visual Basic

```
' comment
x = 1 ' comment
Rem comment
```

### C#

```
// comment
/* multiline
comment */
```

### JScript

```
// comment
/* multiline
comment */
```

## ASSIGNMENT STATEMENTS

### Visual Basic

```
nVal = 7
```

### C#

```
nVal = 7;
```

### JScript

```
nVal = 7;
```

## IF...ELSE STATEMENTS

### Visual Basic

```
If nCnt <= nMax Then
    nTotal += nCnt ' Same as
    nTotal = nTotal + nCnt
    nCnt += 1 ' Same as nCnt
    = nCnt + 1
Else
    nTotal += nCnt
    nCnt -= 1
End If
```

### C#

```
if (nCnt <= nMax)
{
    nTotal += nCnt;
    nCnt++;
}
```

### JScript

```
if(nCnt < nMax) {
    nTotal += nCnt;
    nCnt ++;
}
else {
    nTotal += nCnt;
    nCnt --;
};
```

## CASE STATEMENTS

**Visual Basic**

```
Select Case n
  Case 0
    MsgBox ("Zero")
'   Visual Basic exits the Select at the end of a Case.
  Case 1
    MsgBox ("One")
  Case 2
    MsgBox ("Two")
  Case Else
    MsgBox ("Default")
End Select
```

**C#**

```
switch(n) {
  case 0:
    Console.WriteLine("Zero");
    break;
  case 1:
    Console.WriteLine("One");
    break;
  case 2:
    Console.WriteLine("Two");
    break;
  default:
    Console.WriteLine("?");
}
```

**JScript**

```
switch(n) {
  case 0 :
    Response.Write("Zero");
    break;
  case 1 :
    Response.Write("One");
    break;
  case 2 :
    Response.Write("Two");
  default :
    Response.Write("Default");
}
```

## FOR LOOPS

**Visual Basic**

```
For n = 1 To 10
  MsgBox.Show("The number is " & n)
Next
For Each prop In obj
  prop = 42
Next prop
```

**C#**

```
for (int i = 1; i <= 10; i++)
  Console.WriteLine("The number is {0}", i);
```

**JScript**

```
for (var n = 0; n < 10; n++) {
  Response.Write("The number is " + n);
}
for (prop in obj){
  obj[prop] = 42;
}
```

## WHILE LOOPS

**Visual Basic**

```
While n < 100 ' Test at start of loop
  n += 1 ' Same as n = n + 1
End While '
```

**C#**

```
while (n < 100)
  n++;
```

**JScript**

```
while (n < 100) {
  n++; }
```

## BASICS EXAMPLES

## PARAMETER PASSING BY VALUE

**Visual Basic**

```
Public Sub ABC(ByVal y As Long) ' The argument Y is
    passed by value.
' If ABC changes y, the changes do not affect x.
End Sub
ABC(x) ' Call the procedure
You can force parameters to be passed by value,
    regardless of how they are declared, by enclosing
    the parameters
in extra parentheses.
ABC(x)
```

**C#**

```
// The method:
void ABC(int x)
{
    ...
}
// Calling the method:
ABC(i);
```

**JScript**

```
ABC(i,j);
```

## PARAMETER PASSING BY REFERENCE

**Visual Basic**

```
Public Sub ABC(ByRef y As Long) ' The parameter of ABC
    is declared by reference:
' If ABC changes y, the changes are made to the value of x.
End Sub
ABC(x) ' Call the procedure
```

**C#**

```
// The method:
void ABC(ref int x)
{
    ...
}
// Calling the method:
ABC(ref i);
```

**JScript**

```
N/A (objects (including arrays) are passed by
    reference, but the object to which the variable
    refers to cannot be
changed in the caller). Properties and methods
    change 0d in the callee are visible to the
    caller.
/* Reference parameters are supported for external
    objects, but not internal JScript functions */
comPlusObject.SomeMethod(&foo);
```

## STRUCTURED EXCEPTION HANDLING

**Visual Basic**

```

Try
    If x = 0 Then
        Throw New Exception("x equals zero")
    Else
        Throw New Exception("x does not equal zero")
    End If
Catch
    MessageBox.Show("Error: " & Err.Description)
Finally
    MessageBox.Show("Executing finally block.")
End Try

```

**C#**

```

// try-catch-finally
try
{
    if (x == 0)
        throw new System.Exception ("x equals zero");
    else
        throw new System.Exception ("x does not equal zero");
}
catch (System.Exception err)
{
    System.Console.WriteLine(err.Message);
}
finally
{
    System.Console.WriteLine("executing finally block");
}

```

**JScript**

```

try {
    if (x == 0) {
        throw new Error(513, "x equals zero");
    }
    else {
        throw new Error(514, "x does not equal zero");
    }
}
catch(e) {
    Response.Write("Error number: " + e.number + "<BR>");
    Response.Write("Error description: " + e.message + "<BR>");
}
finally {
    Response.Write("Executing finally block.");
}

```

## SET AN OBJECT REFERENCE TO NOTHING

**Visual Basic**

```
o = Nothing
```

**C#**

```
o = null;
```

**JScript**

```
o = null;
```

# TYPES COMPARISON – VISUAL BASIC, C#, AND JSCRIPT

STORAGE SIZE	VISUAL BASIC	C#	JSCRIPT
Decimal	Decimal (.NET Framework class)	decimal	n/a
Date	Date (.NET Framework class)	Date (.NET Framework class)	n/a. Dates are implemented using JScript's Date object.
(varies)	String (.NET Framework class)	string	String
1 byte	Byte	byte	n/a
2 bytes	Boolean	bool	Boolean
2 bytes	Short, char (Unicode character)	short, char (Unicode character)	n/a
4 bytes	Integer	int	int
8 bytes	Long	long	long
4 bytes	Single	float	float
8 bytes	Double	double	double



# OPERATOR COMPARISON – VISUAL BASIC, C#, AND JSCRIPT

OPERATOR	VISUAL BASIC	C#	JSCRIPT
<b>ADDITIVE</b>			
Addition	+	+	+
Subtraction	-	-	-
<b>MULTIPLICATIVE</b>			
Multiplication	*	*	*
Division	/	/	/
Integer division	\	/	
Modulus (division returning only the remainder)	Mod	%	%
Exponentiation	^	n/a	n/a
<b>ASSIGNMENT</b>			
Assignment	=	=	=
Addition	+=	+=	+=
Subtraction	-=	-=	-=
Multiplication	*=	*=	*=
Division	/=	/=	/=
Integer division	\=	/=	n/a
Concatenate	&=	+=	+=
Modulus	n/a	%=	%=
Left shift	n/a	<<=	<<=
Right shift	n/a	>=	>=
Bitwise-AND	n/a	&=	&=
Bitwise-exclusive-OR	n/a	^=	^=
Bitwise-inclusive-OR	n/a	=	=

# OPERATOR COMPARISON – VISUAL BASIC, C#, AND JSCRIPT

OPERATOR	VISUAL BASIC	C#	JSCRIPT
<b>RELATIONAL AND EQUALITY</b>			
Less than	<	<	<
Less than or equal to	<=	<=	<=
Greater than	>	>	>
Greater than or equal to	>=	>=	>=
Equal	=	==	==
Not equal	<>	!=	!=
Compare two object reference variables	Is	==	n/a
Compare object reference type	TypeOf x Is Class	x is Class	n/a
Compare strings	=	== or String.Equals()	==
Concatenate strings	&	+	+
Shortcircuited Boolean AND	AND	&&	&&
Shortcircuited Boolean OR	OR		
<b>SHIFT</b>			
Left shift	n/a	<<	<<
Right shift	n/a	>	>, >>
<b>SCOPE RESOLUTION</b>			
Scope resolution	.	n/a	n/a

OPERATOR	VISUAL BASIC	C#	JSCRIPT
<b>POSTFIX</b>			
Array element	()	[]	[]
Function call	()	()	()
Type cast	Cint, CDbI, ..., CType	(type)	n/a
Member selection	.	.	.
Postfix increment	n/a	++	++
Postfix decrement	n/a	—	—
<b>UNARY</b>			
Indirection	n/a	* (unsafe mode only)	n/a
Address of	AddressOf	& (unsafe mode only)	n/a
Logical-NOT	Not	!	!
One's complement	BitNot	~	~
Prefix increment	n/a	++	++
Prefix decrement	n/a	—	—
Size of type	n/a	sizeof	n/a
comma	n/a	n/a	,
<b>BITWISE</b>			
Bitwise-AND	BitAnd	&	&
Bitwise-exclusive-OR	BitXor	^	^
Bitwise-inclusive-OR	BitOr		
<b>LOGICAL</b>			
Logical-AND	And	&	&&
Logical-OR	Or		
<b>CONDITIONAL</b>			
Conditional	IIf()	?:	?:
<b>POINTER TO MEMBER</b>			
Pointer to member	n/a	-> (Unsafe mode only)	n/a

## WHAT'S ON THE CD-ROM

The CD-ROM disc included in this book contains many useful files and programs. Before installing any of the programs on the disc, make sure that a newer version of the program is not already

installed on your computer. For information on installing different versions of the same program, contact the program's manufacturer.

### SYSTEM REQUIREMENTS

To use the contents of the CD-ROM, your computer must be equipped with the following hardware and software:

- \* A PC with a Pentium III or faster processor.
- \* Microsoft Windows 2000, Windows NT 4.0 Service Pack 6, or Windows XP Beta 2.
- \* At least 128MB of total RAM installed on your computer.
- \* A double-speed (8x) or faster CD-ROM drive.
- \* A monitor capable of displaying at least 256 colors or grayscale.
- \* A network card.

### AUTHOR'S SOURCE CODE

For *Windows 2000*. The CD provides files that contain all the sample code used throughout this book. You can browse these files directly from the CD-ROM, or you can copy them to your hard drive and use them as the basis for your own projects. To find the files on the CD-ROM, open the D:\RESOURCES\CODE folder. To copy the files to your hard drive, just run the installation program D:\RESOURCES\CODE.EXE. The files will be placed on your hard drive at C:\ProgramFiles\ASPNET. After installing, you can access the files from the START menu. You will need the .NET framework installed on the machine in order to run the samples. See Chapter 1, "Install the .NET Framework," for more information.

### ACROBAT VERSION

The CD-ROM contains an e-version of this book that you can view and search using Adobe Acrobat Reader. You can also use the hyperlinks provided in the text to access all Web pages and Internet references in the book. You cannot print the pages or copy text from the Acrobat files. An evaluation version of Adobe Acrobat Reader is also included on the disc. If you do not currently have Adobe Acrobat Reader 5 installed, the computer will prompt you to install the software.

### INSTALLING AND USING THE SOFTWARE

For your convenience, the software titles appearing on the CD-ROM are listed alphabetically. You can download updates to the software and important links related to the source code at <http://www.threewill.com/authoring/>.

### PROGRAM VERSIONS

Shareware programs are fully functional, free trial versions of copyrighted programs. If you like a particular program, you can register with its author for a nominal fee and receive licenses, enhanced versions, and technical support.

Freeware programs are free, copyrighted games, applications, and utilities. You can copy them to as many computers as you like, but they have no technical support.

GNU software is governed by its own license, which is included inside the folder of the GNU software. There are no restrictions on distribution of this software. See the GNU license for more details.

Trial, demo, or evaluation versions are usually limited either by time or functionality. For example, you may not be able to save projects using these versions.

For your convenience, the software titles on the CD are listed in alphabetical order.

### **Acrobat Reader**

*Freeware.* Acrobat Reader lets you view the online version of this book. For more information on using Adobe Acrobat Reader, see pg. 302.

### **Antechinus C# Programming Editor**

*Shareware.* The Antechinus C# Programming Editor from C Point Pty. Ltd. is an alternate graphic programming environment for creating and testing C# programs. You can find more information at [www.c-point.com](http://www.c-point.com).

### **TextPad**

*Shareware.* TextPad is a general-purpose text editor for many different text files including C# code and HTML code. From Helios Software Solutions, [www.textpad.com](http://www.textpad.com).

### **VMWare Workstation**

*Trial version.* VMWare Workstation lets you create virtual desktop environments on one computer so you can test how your C# programs run in different operating systems. From VMWare, [www.vmware.com](http://www.vmware.com).

### **XML Spy IDE**

*Evaluation version.* XML Spy IDE v3.5 is a development environment that supports working with XML. This includes editing and validating XML documents, editing and validating using Schemas or DTDs, and editing and transforming stylesheets. From ALTOVA, [www.xmlspy.com](http://www.xmlspy.com).

## **TROUBLESHOOTING**

We tried our best to compile programs that work on most computers with the minimum system requirements. Your computer, however, may differ and some programs may not work properly for some reason.

The two most likely problems are that you don't have enough memory (RAM) for the programs you want to use, or you have other programs running that are affecting installation or running of a program. If you get error messages like `Not enough memory` or `Setup cannot continue`, try one or more of these methods and then try using the software again:

- \* Turn off any anti-virus software.
- \* Close all running programs.
- \* In Windows, close the CD-ROM interface and run demos or installations directly from Windows Explorer.
- \* Have your local computer store add more RAM to your computer.

If you still have trouble installing the items from the CD-ROM, please call the Hungry Minds Customer Service phone number: 800-762-2974 (outside the U.S.: 317-572-3994).

# USING THE E-VERSION OF THE BOOK

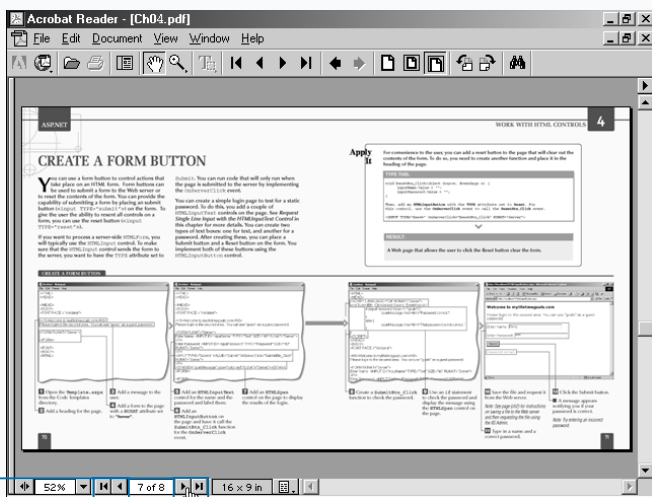
You can view *ASP.NET: Your visual blueprint for creating Web applications on the .NET framework* on your screen using the CD-ROM disc included at the back of this book. The CD-ROM disc allows you to search the contents of each chapter of the book for a specific word or phrase. The CD-ROM disc also provides a convenient way of keeping the book handy while traveling.

You must install Adobe Acrobat Reader on your computer before you can view the book on the

CD-ROM disc. This program is provided on the disc. Acrobat Reader allows you to view Portable Document Format (PDF) files, which can display books and magazines on your screen exactly as they appear in printed form.

To view the contents of the book using Acrobat Reader, display the contents of the disc. Double-click the PDFs folder to display the contents of the folder. In the window that appears, double-click the icon for the chapter of the book you want to review.

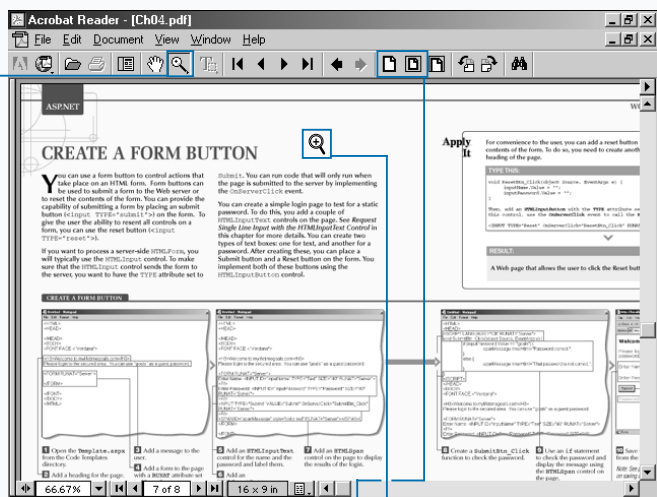
## USING THE E-VERSION OF THE BOOK



### FLIP THROUGH PAGES

1 Click one of these options to flip through the pages of a section.

- First page
- Previous page
- Next page
- Last page



### ZOOM IN

1 Click to magnify an area of the page.

2 Click the area of the page you want to magnify.

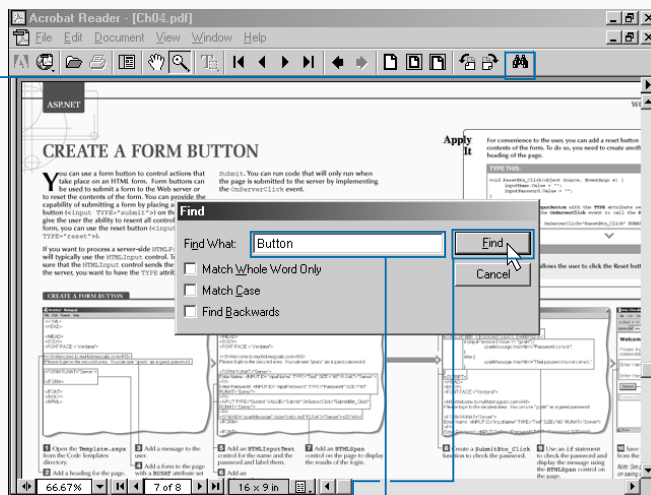
3 Click one of these options to display the page at 100% magnification () or to fit the entire page inside the window (.

## Extra

To install Acrobat Reader, insert the CD-ROM disc into a drive. In the screen that appears, click Software. Click Acrobat Reader and then click Install at the bottom of the screen. Then follow the instructions on your screen to install the program.

You can make searching the book more convenient by copying the .pdf files to your own computer. Display the contents of the CD-ROM disc and then copy the PDFs folder from the CD to your hard drive. This allows you to easily access the contents of the book at any time.

Acrobat Reader is a popular and useful program. There are many files available on the Web that are designed to be viewed using Acrobat Reader. Look for files with the .pdf extension. For more information about Acrobat Reader, visit the Web site at [www.adobe.com/products/acrobat/readermain.html](http://www.adobe.com/products/acrobat/readermain.html).



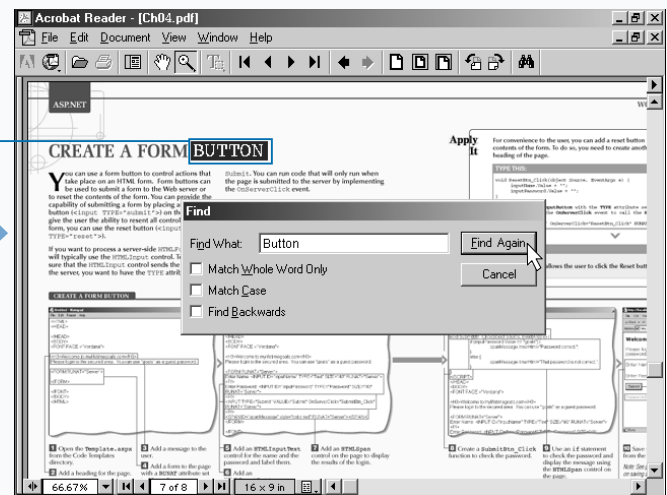
### FIND TEXT

1 Click  to search for text in the section.

■ The Find dialog box appears.

2 Type the text you want to find.

3 Click Find to start the search.



■ The first instance of the text is highlighted.

■ Repeat Steps 1 and 3 to find the next instance of the text.

## HUNGRY MINDS, INC. END-USER LICENSE AGREEMENT

**READ THIS.** You should carefully read these terms and conditions before opening the software packet(s) included with this book ("Book"). This is a license agreement ("Agreement") between you and Hungry Minds, Inc. ("HMI"). By opening the accompanying software packet(s), you acknowledge that you have read and accept the following terms and conditions. If you do not agree and do not want to be bound by such terms and conditions, promptly return the Book and the unopened software packet(s) to the place you obtained them for a full refund.

1. **License Grant.** HMI grants to you (either an individual or entity) a nonexclusive license to use one copy of the enclosed software program(s) (collectively, the "Software") solely for your own personal or business purposes on a single computer (whether a standard computer or a workstation component of a multi-user network). The Software is in use on a computer when it is loaded into temporary memory (RAM) or installed into permanent memory (hard disk, CD-ROM, or other storage device). HMI reserves all rights not expressly granted herein.

2. **Ownership.** HMI is the owner of all right, title, and interest, including copyright, in and to the compilation of the Software recorded on the disk(s) or CD-ROM ("Software Media"). Copyright to the individual programs recorded on the Software Media is owned by the author or other authorized copyright owner of each program. Ownership of the Software and all proprietary rights relating thereto remain with HMI and its licensors.

3. **Restrictions On Use and Transfer.**

(a) You may only (i) make one copy of the Software for backup or archival purposes, or (ii) transfer the Software to a single hard disk, provided that you keep

the original for backup or archival purposes. You may not (i) rent or lease the Software, (ii) copy or reproduce the Software through a LAN or other network system or through any computer subscriber system or bulletin-board system, or (iii) modify, adapt, or create derivative works based on the Software.

(b) You may not reverse engineer, decompile, or disassemble the Software. You may transfer the Software and user documentation on a permanent basis, provided that the transferee agrees to accept the terms and conditions of this Agreement and you retain no copies. If the Software is an update or has been updated, any transfer must include the most recent update and all prior versions.

4. **Restrictions on Use of Individual Programs.** You must follow the individual requirements and restrictions detailed for each individual program in the What's on the CD-ROM appendix of this Book. These limitations are also contained in the individual license agreements recorded on the Software Media. These limitations may include a requirement that after using the program for a specified period of time, the user must pay a registration fee or discontinue use. By opening the Software packet(s), you will be agreeing to abide by the licenses and restrictions for these individual programs that are detailed in the What's on the CD-ROM appendix and on the Software Media. None of the material on this Software Media or listed in this Book may ever be redistributed, in original or modified form, for commercial purposes.

5. **Limited Warranty.**

(a) HMI warrants that the Software and Software Media are free from defects in materials and workmanship under normal use for a period of sixty (60) days from the date of purchase of this Book. If HMI receives notification within the warranty period of defects in materials or workmanship, HMI will replace the defective Software Media.



**(b) HMI AND THE AUTHOR OF THE BOOK DISCLAIM ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WITH RESPECT TO THE SOFTWARE, THE PROGRAMS, THE SOURCE CODE CONTAINED THEREIN, AND/OR THE TECHNIQUES DESCRIBED IN THIS BOOK. HMI DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE SOFTWARE WILL BE ERROR FREE.**

(c) This limited warranty gives you specific legal rights, and you may have other rights that vary from jurisdiction to jurisdiction.

## **6. Remedies.**

(a) HMI's entire liability and your exclusive remedy for defects in materials and workmanship shall be limited to replacement of the Software Media, which may be returned to HMI with a copy of your receipt at the following address: Software Media Fulfillment Department, Attn.: *ASP.NET: Your visual blueprint for creating Web applications on the .NET framework*, Hungry Minds, Inc., 10475 Crosspoint Blvd., Indianapolis, IN 46256, or call 1-800-762-2974. Please allow four to six weeks for delivery. This Limited Warranty is void if failure of the Software Media has resulted from accident, abuse, or misapplication. Any replacement Software Media will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer.

(b) In no event shall HMI or the author be liable for any damages whatsoever (including without limitation damages for loss of business profits, business interruption, loss of business information, or any other

pecuniary loss) arising from the use of or inability to use the Book or the Software, even if HMI has been advised of the possibility of such damages.

(c) Because some jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation or exclusion may not apply to you.

**7. U.S. Government Restricted Rights.** Use, duplication, or disclosure of the Software for or on behalf of the United States of America, its agencies and/or instrumentalities (the "U.S. Government") is subject to restrictions as stated in paragraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause of DFARS 252.227-7013, or subparagraphs (c) (1) and (2) of the Commercial Computer Software - Restricted Rights clause at FAR 52.227-19, and in similar clauses in the NASA FAR supplement, as applicable.

**8. General.** This Agreement constitutes the entire understanding of the parties and revokes and supersedes all prior agreements, oral or written, between them and may not be modified or amended except in a writing signed by both parties hereto that specifically refers to this Agreement. This Agreement shall take precedence over any other documents that may be in conflict herewith. If any one or more provisions contained in this Agreement are held by any court or tribunal to be invalid, illegal, or otherwise unenforceable, each and every other provision shall remain in full force and effect.

## Symbols and Numbers

---

- \*.asax. See Global .asax
- \*.asmx, 152, 154, 172
- \*.aspx, 205
- \*.cs, 152
- /bin directory, 152, 176, 184
  - business component, 188
  - DLL, 178
  - Web site, 179

## A

---

- Active Directory, 258
- Active Server Page (ASPX), 150
- ADO.NET, 147, 184
- Adobe Acrobat, 300
  - Reader, 301
- advertisement banners
  - displaying, 114–115
  - elements of, 115
  - generating income, 114
- aliases, 177
- anchor tags, 80–81, 106
- anonymous access, 254
- Antechinus C# Programming Editor, 301
- array
  - creating, 108
  - goals, 159
  - index, 63
  - initializing, 108
  - list, 129, 219
  - single-dimension, 56
  - values, 79
  - Web Method, 158
  - Web services, 158–159
  - XML, 158
- ASMX file, 152, 154
- ASPX (Active Server Page), 150
- Authentication
  - Basic, 252–253
  - cookies, 256
  - definition of, 255
  - Forms, 256–257, 259
    - Active Directory, 258
    - CD-ROM, 257
    - HTTP, 258
    - Internet Services Manager (ISM), 256
  - login, 257
  - password, 258
  - SQL, 258
  - Windows 2000, 258
  - user credentials, 255
  - Windows, 252–257, 260
- author's source code, 300–301
- authorization, 260–261



## B

---

- banners, 114–115
- batch files, 177
- Beta 1, 7
- Beta 2, 7
- binding
  - data, 140, 279
  - datagrid, 249
- blocks, 276–277
  - C#, 276
  - Visual Basic, 276
- Boolean
  - expressions, 46, 50
  - input, 77
  - request, 98–99
- boundaries, 120–121
- box
  - check, 27, 91, 98–99
  - drop-down list, 82, 116–117
  - list, 79
  - password, 96
  - text, 27, 91, 110, 116, 119, 122
    - creating, 97
    - hiding, 97
    - type of, 96
- business
  - layer, 189–191
  - logic, 176, 188
- button
  - controls, 92–93
  - definition of, 91
  - form, 70–71
  - graphical, 73, 95
  - HTML 4.0, 93
  - hyperlink, 91, 94
  - image, 73, 91, 95
  - radio, 27, 78, 91, 100–101
  - representing columns, 145
  - submit, 27, 103, 135, 137, 139, 143

## C

### C#, 34–65, 213, 287

- Antechinus C# Programming Editor, 301
- assignment statements, 292
- case sensitive, 35
- case statements, 293
- class file, 34
- code, 38–39
- comments, 292
- compiler, 36–37
- components, 177, 188
- concatenate strings, 52–53
- examples of, 292
- exception handling, 295
- file, 34, 37
- FOR Loops, 293
- handling, 62
- if...else statements, 292
- namespace aliases, 34
- object reference, 295
- operators, 297–299
- parameters, 294
- Request.Form, 194
- runtime error, 41–42
- script blocks, 276
- source file, 176, 180, 184
- statement, 63
- text editor, 34
- types, 296
- variables, 40–43
- WHILE Loops, 293
- writing applications, 34–35

### cache

- data, 209
- expiration, 209
- pages, 206–207

### calendar

- control, 105, 121
- CultureInfo, 265
- dates, 104–105
- Web server, 121

### CD-ROM, 3, 24

- batch files, 177
- contents of, 300–301
- Forms Authentication, 257
- Globally Unique Identifier (GUID), 89

- table cell example, 109
- Web services parameters, 157
- Windows Authentication, 253

### check box, 27, 91

CLR (Common Language Runtime), 3, 274

CLS (Common Language Specification), 274–275, 283

code

- application, 213
- business services, 2
- cache, 179
- client-side, 92
- Code-behind, 192
- commenting, 38
- components, 65
- conditional statements, 46
- creating, 176, 180, 186
- data access, 136, 184, 190
- delimiters, 277
- embedded, 50
- encapsulating, 65
- errors, 62
- event handler, 33, 99
- formatting, 38–39
- full source, 187
- Global.asax, 221
- HTML, 28
- insert, 137
- label control, 110
- managed, 178
- methods, 58, 60
- migrating, 64
- modifying, 110
- optimizing, 48
- presentation, 184, 186
- server-side, 28, 33, 132, 141–142, 192, 196, 276
- session, 213
- structuring, 48
- templates, 32, 73, 108, 130
- tracking, 249
- transactional, 190
- Unicode, 262
- validation, 183
- Web application, 186
- writing, 190

Code-behind, 25, 192, 193

collections, 225

cookies, 248

definition of, 56

enumerating, 56

headers, 248

loop, 108

Common Language Runtime (CLR), 3, 274

Common Language Specification (CLS), 274–275, 283

Common Type System (CTS), 42

compiler, 36–37

components

.COM, 179

.NET, 179

business, 188, 190

control, 183

creating simple, 179

data access, 186

definition of, 176, 178

multiple, 179

simple, 176–178

stateful, 180–183

stateless, 180

Web page, 182

Web service, 182

Computer Management Console, 23

conditional statements, 46

configuration, 3, 17–18, 25

application, 236

custom, 240–241

events, 32

HTML, 194

session state, 227

setting, 238–239

tracing, 250

URL, 213

connection

object, 134

string, 185

console

application conversions, 64–65

client, 174–175

container control, 111–113

control

ActiveX, 202

bound, 134

calendar, 121

container, 111–113

creating, 111

data, 127, 132, 190

DataGrid, 127, 140–141, 144

DataList, 127

declaring, 118

editing, 132

hiding specific group, 111

HTML, 66–67

naming, 110–111

page, 132, 268–269

providing a container, 111

Repeater, 127–129

Server, 2

server-side, 30, 77, 279

sorting data, 132

specifying, 118

stateful components, 183

TableCell, 108

TableRow, 108

tree, 248

two equal values, 119

using placeholder, 111

validation, 118, 120, 122, 124

Web page, 134

Control Panel, 4, 14, 18, 20–23

conversions, 55

console application, 64

explicit, 54, 63

implicit, 54

cookies, 207

authentication, 261

collection, 248

creating, 230–233

definition of, 211

expiration date, 231

Forms Authentication, 256

Internet Information Server (IIS) MMC, 230

subkeys, 232–233

Web browser, 231

credentials, 258–259

CTS (Common Type System), 42

culture, 264, 265, 266, 267, 268, 269, 272

CultureInfo, 264, 265

custom

configuration, 240–241

error handling, 244–246

modules, 211

settings, 240–241

customer order system, 144

## D

**data.** *See also* [data access](#)

- alternating rows, 128
  - binding, 129, 132, 140, 279
  - cache, 208
  - checking, 120
  - control, 133
  - defining layout, 128
  - deleting, 138–139
  - display, 127–129, 144, 186, 190, 196
  - double-type, 120
  - editing control, 132
  - enhance viewing, 133
  - filtering, 142
  - footers, 128
  - formatting, 130
  - headers, 128
  - inserting, 134
  - integer-type, 120
  - layer, 188–189
  - list, 187
  - posting, 92
  - reading, 194
  - repeating, 128
  - separate rows, 128
  - shared, 166
  - sorting, 132, 140–141
  - SQL, 132–137
  - stores, 146, 190, 208
  - string, 120
  - transporting, 146
  - types, 42–43
- data access,** [2](#), [166](#)
- code, 136, 184, 190
  - command, 126–127
  - components, 186
  - connection, 126
  - controls, 126
  - DataAdapter, 127
  - DataGrid, 127
  - DataList, 127
  - definition of, 126
  - disconnected, 147
  - introduction to, 126
  - layer, 188–189
  - Repeater, 127

**database**

- accessing, 3
  - establishing connection, 134
  - inserting data, 134
- DCOM (Distributed Component Object Model),** [150](#)
- debugging,** [239](#), [248](#)
- page-level, 242–243
- default document,** [14](#)
- delete**
- data, 138
  - SQL database, 139
  - Web page, 200
- delimiters,** [277](#)
- desktop shortcuts,** [21](#)
- DHTML,** [72](#)
- directives**
- application, 216–217
  - assembly, 216–217
  - Global.asax, 216–217
  - import, 216
  - page, 216, 281, 284
  - processing, 218
- directory.** *See also* [Active Directory](#)
- default**
- Internet Information Server (IIS), [14](#)
  - home, [8](#), [9](#), [10](#), [13](#)
  - Internet Information Server (IIS), [12](#)
  - removing, [11](#)
  - root, [250](#)
  - virtual, [10–13](#), [22](#), [23](#)
  - web, [20](#)
- Disco (Web Services Discovery),** [150](#), [174](#)
- Distributed Component Object Model (DCOM),** [150](#)
- DIV,** [198](#)
- DLL**
- /bin directory, [178](#)
  - source code, [176](#)
- drop-down list,** [27](#), [79](#), [82](#), [116](#)
- input, [102–103](#)
  - validating, [117](#)

## E

- e-mail,** [5](#)
- sending, [204–205](#)
  - SMTP server, [204](#)
  - text format, [205](#)
- ECMAScript,** [287](#)

## encoding

- attributes, 263
- file, 262
- languages, 262
- response, 262
- UCS Transformation Format (UTF-8), 262

## enumeration

- constant value, 161
- creating, 160

## error, 62

- custom handling, 244–246
- details, 242–243, 247
- handling programmatically, 246, 247
- message, 116, 239
- query, 245
- stored procedure, 142
- summarizing validation, 124
- troubleshooting, 244
- validation, 124
- Web page, 200
- Web server, 238, 239

## event, 45

- application, 220
- handler, 32–33, 64, 93–94, 99, 104, 192–193, 212, 214, 224, 226
  - Global.asax, 220–221
- procedure, 140
- server-side, 106

## exceptions, 63

## expressions

- regular, 122–123
- validating, 122–123

## F

### File Transfer Protocol (FTP) Server, 5

## files

- batch, 177
- converting, 273
- copying into Web server, 178
- describing data, 146
- name, 14, 24, 25
  - C#, 37
- nonbinary text, 146
- saving, 25
- sending, 88
- uploading, 88–89
- XML, 114

## flag, 138

## footer, 130

## form

- adding controls, 111
- attributes, 69
- button, 70–71
- comparing two fields, 118
- event handler, 193
- hidden information, 86
- hidden variable, 87
- HTML, 29, 32
- image, 95
- processing, 122
- registration, 77
- resubmitting, 122, 124
- server, 76, 100, 120, 131, 133, 141
- validation, 122, 124
  - Web, 2, 30–31, 134

### Framework SDK, 150

### FTP (File Transfer Protocol) Server, 5

## G

### GAC (Global Assembly Caches), 178

## garbage, 182

## GET, 152

## global

- assemblies, 178
- variables, 277

### Global Assembly Caches (GAC), 178

## Global.asax, 211

- application, 218
- code, 221
- creating file, 212–215
- event handlers, 220–221
- parameters, 221
- processing directives, 216–217
- server-side objects, 218–219
- session, 218
- statistics, 215

### Globally Unique Identifier (GUID), 89

## goals, 225

- array, 159
- classifying, 160
- identifier, 162
- list, 165
- returning, 165

graphics  
   creating button, 95  
   drop-down list box, 82  
   Web browser, 107

greeting  
   displaying, 173  
   personal, 181

GUID (Globally Unique Identifier), 89

## H

help, 107  
 hidden information, 86  
 home directory, 8–10, 13  
 HTML, 150  
   4.0, 72–73  
   4.0 button, 93  
   anchor tags, 81  
   code, 28  
   configuration, 194  
   container controls, 67  
   controls, 66–67, 102  
   embedded, 278  
   footer, 15  
   form, 25–27, 29, 32  
   formatting files, 205  
   formatting XML, 148  
   input controls, 67  
   Image Control, 67  
   markup, 26, 274  
   page, 28  
   server controls, 66–67, 90, 98, 108, 288–291  
   simple page, 196  
   specifying horizontal line, 128  
   table, 132–133  
   tags, 2, 26, 198  
   transforming XML, 149  
   two-tier web form, 184  
   user interface, 64  
 HTTP, 150, 151, 152. *See also* GET; POST; SOAP  
   application, 211  
   Forms Authentication, 258  
   request, 69  
   response packets, 200  
   values, 194  
   Windows Authentication, 252  
 hyperlink, 91, 145  
   creating, 94, 106  
   trace, 250  
   Web browser, 106

## I

IIS (Internet Information Server), 10–11, 13, 17, 210, 228  
   default directory, 14  
   installing, 4–5  
   log file, 16  
   Microsoft Management Console (MMC), 210, 230  
   virtual directory, 12  
   Windows Authentication, 252, 260  
 IL (Intermediate Language), 274  
 image, 91  
   aligning, 82  
   border, 83  
   button, 91, 95  
   rendering, 82–83, 107  
 impersonation, 254–255  
 input  
   Boolean, 77  
   checking boundaries, 120–121  
   drop-down list, 79, 102–103  
   login name, 116  
   multiple line, 76  
   parameters, 150  
   request text, 96  
   single line, 74–75  
   soliciting, 102  
   text, 97  
   validating, 122  
 insert  
   code, 137  
   parameters, 139  
 installing, 4–7  
   IIS (Internet Information Server), 4–5  
   .NET Framework, 6–7  
 Intermediate Language (IL), 274  
 Internet Explorer, 154, 252  
 Internet Information Server (IIS), 10–11, 13, 17, 210, 228  
   default directory, 14  
   installing, 4–5  
   log file, 16  
   Microsoft Management Console (MMC), 210, 230  
   virtual directory, 12  
   Windows Authentication, 252, 260  
 Internet Information Service (IIS), 20  
 Internet Services Manager (ISM), 5, 10, 16, 18, 20  
   browsing, 21  
   exploring, 22  
   Forms Authentication, 256  
   Windows Authentication, 252, 254

ISM (Internet Services Manager), 5, 10, 16, 18, 20  
iterative statements, 50

## JScript, 287

- assignment statements, 292
- case statements, 293
- comments, 292
- examples of, 292
- exception handling, 295
- FOR Loops, 293
- if...else statements, 292
- migrating, 286–287
- object reference, 295
- operators, 297–299
- parameters, 294
- types, 296
- WHILE Loops, 293

JScript.NET, 286–287

## L

### language, 30

- C#, 2
- client-side, 274
- CLR (Common Language Runtime), 274
- CLS (Common Language Specification), 274–275
- compliant, 176
- cross-language compatibility, 42
- encoding, 262
- first class, 283
- IL (Intermediate Language), 274
- industrial strength, 287
- multiple, 272, 274–275
- object-oriented, 45, 60
- pattern-matching, 122
- runtime, 179
- server-side, 274
- specifying, 213
- support, 2
- Visual Basic, 2, 54
- Web page, 275
- Web services, 152

### layers

- business, 188–191
- data, 188–189
- presentation, 188
- user interface, 189

### link

- button, 91

### links. *See also* hyperlink

- creating, 80–81, 106
- navigating, 80

### lists

- array, 219
- box, 79
- comma-separated, 129
- custom, 128
- data-bound, 187
- displaying complex, 130–131
- drop-down, 79
- master, 144–145
- specifying direction, 130

### literals, 109

### localize, 268, 269

### log, 16, 17

- enable, 16
- file, 17
- Internet Information Server (IIS), 16

### login

- Forms Authentication, 257
- input name, 116
- page, 96

### Loops

- collections, 108
- FOR, 293
- WHILE, 293

## M

### machine-wide code, 179

### master list, 144–145

### master-detail relationships, 144–145

### MDAC (Microsoft Data Access Components), 6

### memory, 63, 301

### message

- displaying, 122, 124
- header, 124
- string, 176
- summary, 124
- validation, 118, 120, 123

### methods, 45

- code, 58, 60
- creation, 221
- declaring, 58, 60
- definition of, 58, 60
- destruction, 221
- input parameters, 150



- objects, 59
- public, 152
- request, 196
- signature, 60–61
- using, 58, 60
- Web service, 156

Microsoft Data Access Components (MDAC), 6

Microsoft Management Console (MMC), 210

Microsoft Message Queue (MSMQ), 237

migrating, 280–281

- code, 64

- JScript, 286–287

- VB.NET Syntax, 284–285

- Visual Basic, 282–285

MMC (Microsoft Management Console), 210

Mobile application, 275

mode, 244

modules, 211

MSDN

- culture, 269

MSMQ (Microsoft Message Queue), 237

Multiple Threaded Apartment (MTA), 284

## N

namespace

- aliases, 177

- importing, 204

- objects, 166

- properties, 44

- system, 176

native serialization, 147

Netscape, 287

notation, 42

Notepad, 2, 34

## O

object, 58

- application, 170, 210

- collection, 108

- command, 134

- connection, 134

- constructors, 183

- creating new object, 162

- DataSet, 147

- methods, 59

- multiple, 218

- namespace, 166

- properties, 162

- reference setting, 295

- server-side, 211, 218–219

- session, 168, 182, 210–211, 227

- single, 218

- SQLConnection, 136, 138

- Web Method, 168

- Web service, 162, 168

operators

- arithmetic, 52, 63

- assignment, 52

- comparison of, 297–299

output

- formatting, 128

- formatting with templates, 130

- specifying, 130

## P

page

- directives, 280, 284

- localize, 268–269

- state, 234–235

- trace, 248–249

panels, 91

- hiding, 112

parameters

- client console, 175

- command-line, 64

- Global.asax, 220

- input, 150

- insert form, 139

- passing by reference, 294

- passing by value, 294

- passing data, 142

- reading, 137

- title page, 185

- value, 154

- Visual Basic, 285

- Web service, 163

- Web services, 156–157

password

- box, 27, 96

- Forms Authentication, 258

- impersonation, 255

- specifying, 134

- static, 96

performance, 3

permissions, 12

placeholder  
tag, 198  
using for controls, 111

pop-up  
help text, 107  
menu, 21

port access, 150

POST, 151–152  
testing, 155

postback, 30

power, 3

procedures  
error, 142  
stored, 142–143

program  
object-oriented, 35  
versions, 300

properties, 60  
access, 44  
definition of, 44–45  
image, 82  
namespace, 44

proxy class, 172, 174–175

## Q

queries  
ad hoc, 142  
control, 142  
error, 245  
string, 195, 199, 201, 206, 245  
XML engine, 164

## R

radio buttons, 27, 100–107

ranges  
currency, 120  
dates, 120  
double-type data, 120  
integer-type data, 120  
string data, 120

RDBMS (Relational Database Management System), 126

redirecting, 200–201

RegionInfo, 266–267

regular expressions, 122–123

Relational Database Management System (RDBMS), 126

Remote Procedure Calls (RPC), 150

render functions, 278–279  
images, 107  
troubleshooting, 279

Request.Form, 194–195

Request.Params, 196–197

Resource Manager, 270–273

Response.Redirect, 200–201

Response.Write, 198–199

return value, 150

rollover effect, 95

root directory, 250

runtime, 7, 179  
error, 41–42

## S

scalability, 224

script  
blocks, 187, 276–277  
delimiters, 216  
external file, 277  
server-side, 187, 191

Secured Sockets Layer (SSL), 252, 258

security, 3, 23, 210  
applications, 255  
authorization, 260  
impersonation, 254–255

server, 2  
controls, 2  
web, 2

session  
code, 213  
color, 229  
disable, 170  
end, 214  
Global.asax, 218  
in-process, 228  
object, 168–169, 182, 211, 227  
out-of-process, 228  
SQL server, 228  
start, 214  
state, 226–229  
types of, 168  
variables, 226, 229

settings  
application, 236–237  
custom, 240–241

signature, 60–61

simple components

creating, 176–179

Single Threaded Apartment (STA), 284

SMTP server e-mail, 204

snap-in, 3

snap-ins, 5

SOAP, 150–152

software

installing, 300

using, 300

sort

ascending (ASC), 140

data, 140–141

descending (DESC), 140

SPAN, 198

SQL

data, 132, 144

data stores, 166

Delete SQL string, 139

deleting, 138–139

displaying data, 133

Forms Authentication, 258

inserting, 134–135

select statements, 184

sending custom statement, 136

server, 147, 228

sorting, 140–141

statement, 134, 166

statements, 142

string, 136

updating, 137

updating data, 136

XML query engine, 164

statistics, 215

stored procedures

execute, 142–143

stores

data, 190

trace, 251

string

appending, 52

building, 180

concatenate, 52–53

connection, 185

Delete SQL, 139

insert SQL, 135

keys, 194

message, 176

query, 195, 199, 201, 206, 245

SQL, 137–138

values, 194

submit button, 27, 135, 137, 139, 143

synchronization, 224

syntax

Global.asax, 290

web.config, 290–291

**T**

table

building, 84–85, 87, 108–109

cell, 91, 109

colors, 85, 130

definition of, 91

HTML, 132–133

literals, 109

properties of, 84

row, 91

tags

anchor, 80–81, 106

attributes, 96

DIV, 198

HTML, 2, 198

HTML Input Submit Button, 92

placeholder, 198

range validator, 120

single, 260

SPAN, 198

templates

code, 24, 27–28, 32, 108, 130

definition of, 24

file, 24

generic, 24

Repeater control, 128

text

area, 27

box, 27, 74–76, 91, 96–97, 110, 116, 119, 122

command, 147

format, 205

input, 96

manipulating, 110

request input, 97

validating, 122

viewer, 146

TextPad, 301

three-tier Web form, 188–191

title page, 185

tools

administrative, 8, 10, 14, 20, 22  
debugging, 65

trace

application-level, 250–251  
configuration, 250  
hyperlink, 250  
information, 248–249  
page-level, 248–250  
storage, 251  
virtual directory, 250  
Web site, 250

troubleshooting, 17

CD-ROM, 301  
custom errors, 244  
render functions, 279

two-tier web form, 184–187

## U

---

UCS Transformation Format (UTF-8), 262, 272

Unicode, 262

URL

configuration, 213  
impersonation, 254  
passing data, 195  
Web service, 172

user

authorization, 260–261  
credentials, 255  
ID, 134

## V

---

validation

based on range of values, 120  
compare fields for, 118  
enable basic process of, 116  
summarize errors from, 124  
use regular expressions for, 122

variables

application, 170–171, 224–225  
converting, 54  
declaring, 40–41, 48, 292  
defining, 40–41  
displaying, 197  
fixed values, 160  
form, 197

global, 277

hidden, 87

hidden information, 86

initialization, 40–43, 48, 180

initializing, 108, 286

iteration, 56

long, 55

private, 180

public, 180

server, 248

session, 226, 229

setting, 214

updating, 223

VB.NET Syntax, 282–283, 285

migrating, 284

VBScript. *See* Visual Basic

virtual directory

application, 210

trace, 250

Visual Basic, 54, 213, 287

assignment statements, 292

case statements, 293

comments, 292

examples of, 292

exception handling, 295

FOR Loops, 293

if...else statements, 292

migrating, 282–285

object reference, 295

operators, 297–299

parameters, 285

passing by reference, 294

passing by value, 294

Request.Form, 194

script blocks, 276

types, 296

WHILE Loops, 293

Visual Studio.NET (VS.NET), 6, 21

VMware Workstation, 301

VS.NET (Visual Studio.NET), 6, 21

## W

---

W3C, 146

Web browser, 72, 88

cookie, 231

graphics, 107

- hyperlink, 106
- image, 73, 82
- testing, 155
- types, 202, 203
- Web class, 172, 174**
- Web controls, 95–96, 109, 234**
  - basic, 91
  - classification, 90
  - data list, 91
  - introduction, 90–93
  - list, 91
  - rich, 91
- Web forms, 2, 25–26, 30, 33, 186**
  - control, 92
  - controls, 90
  - creating, 31
  - server-side form, 192
- Web Method, 154, 160**
  - array, 158–159
  - creating, 166
  - default definition, 170
  - return, 162
  - session object, 168
- Web page**
  - adding advertisement banners, 114
  - breaking into sections, 111
  - Code-behind, 193
  - components, 182
  - creating, 187
  - custom testing, 155
  - deleting, 138, 200
  - error, 200
  - form control, 134
  - inserting SQL data, 134
  - viewing multiple pages, 112
- Web server, 2, 8–10, 14, 24–25, 66, 80, 88, 108**
  - calendar, 104–105, 121
  - controls, 90, 99–107, 288–290
  - copying files, 178
  - error, 238
  - processing events, 32
  - request, 28
  - response, 28
  - restarting, 19
  - starting, 19
  - stopping, 19
  - XML files, 146
- Web service, 2, 25, 275**
  - application object, 170–171
  - benefits, 150
  - clients, 150, 152, 156, 162, 164
  - components, 182
  - creating, 152, 153
  - creating client console, 174, 175
  - creating client page, 172–173
  - DataSet, 167
  - enumerated type, 160–161
  - figure of, 151
  - introduction, 150
  - language, 152
  - method, 156
  - parameters, 156–157, 163
  - protocols, 151
  - return SQL data, 166–167
  - return XML, 164–165
  - returning an array, 158–159
  - returning an object, 162–163
  - session object, 168–169
  - simple, 152
  - standards, 151
  - testing, 154–155
  - URL, 172
  - writing, 152
  - XML, 151, 165
- Web Services Description Language (WSDL), 150**
- Web Services Discovery (Disco), 150, 174**
- Web site**
  - /bin directory, 179
  - browsing, 20–21
  - creating, 26
  - default, 21, 23, 25, 27, 29, 31, 33
  - displaying text, 110
  - exploring, 22
  - interactive, 28
  - moving, 178
  - pausing, 18–19
  - properties page, 15
  - starting, 18–19
  - stopping, 18–19
  - trace, 250

## Windows

2000, 204, 254, 258, 260

Authentication, 252–256

CD-ROM, 253

Internet Explorer, 252

Internet Information System (IIS), 252, 260

Internet Services Manager (ISM), 252, 254

Explorer, 11, 22–23

Form

application, 275

Wireless Markup Language (WML), 148

Wizard, 5, 21

Virtual Directory Creation, 11, 13

Windows Component, 4

WML (Wireless Markup Language), 148

WSDL (Web Services Description Language), 150

## X

---

### XML

array strings, 158

code, 149

data sources, 146–147

definition of, 236

display, 148–149

file, 114

format, 273

loading documents, 165

request display, 154

Spy IDE, 301

transform, 148–149

Web service, 151, 164

XSL, 148

XSLT, 164



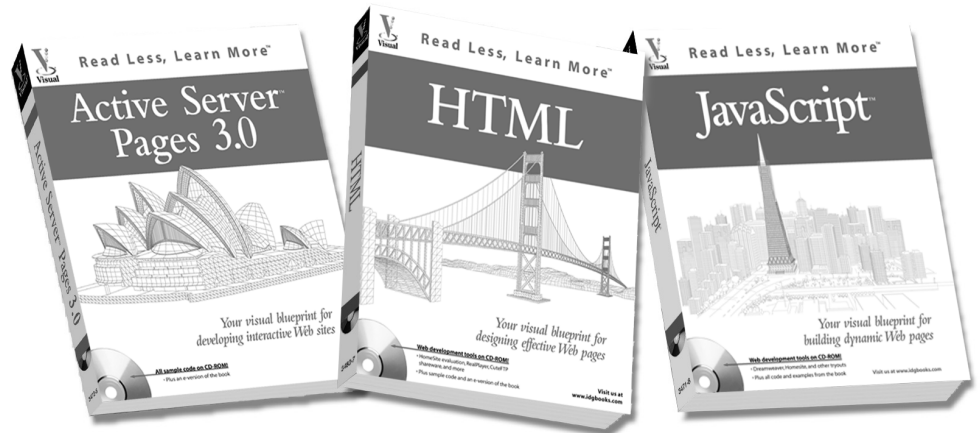


# Read Less – Learn More™

## New Series!

*The visual alternative to learning complex computer topics*

*For experienced computer users, developers, network professionals who learn best visually.*



**Extra  
Apply  
It**

“Apply It” and “Extra” provide ready-to-run code and useful tips.

Title	ISBN	Price
Active Server™ Pages 3.0: Your visual blueprint for developing interactive Web sites	0-7645-3472-6	\$26.99
HTML: Your visual blueprint for designing effective Web pages	0-7645-3471-8	\$26.99
Java™: Your visual blueprint for building portable Java programs	0-7645-3543-9	\$26.99
JavaScript™: Your visual blueprint for building dynamic Web pages	0-7645-4730-5	\$26.99
JavaServer™ Pages: Your visual blueprint for designing dynamic content with JSP	0-7645-3542-0	\$26.99
Linux®: Your visual blueprint to the Linux platform	0-7645-3481-5	\$26.99
Perl: Your visual blueprint for building Perl scripts	0-7645-3478-5	\$26.99
PHP: Your visual blueprint for creating open source, server-side content	0-7645-3561-7	\$26.99
Unix®: Your visual blueprint to the universe of Unix	0-7645-3480-7	\$26.99
XML: Your visual blueprint for building expert Web pages	0-7645-3477-7	\$26.99

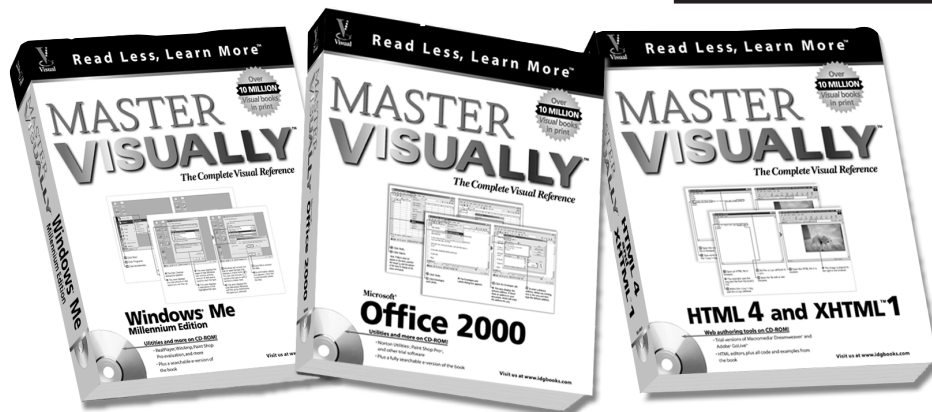
**Over 10 million Visual books in print!**



with these two-color Visual™ guides

*The Complete Visual Reference*

MASTER  
VISUALLY™



For visual learners who want an all-in-one reference/tutorial that delivers more in-depth information about a technology topic.



“Master It” tips provide additional topic coverage

Title	ISBN	Price
Master Active Directory™ VISUALLY™	0-7645-3425-4	\$39.99
Master Microsoft® Access 2000 VISUALLY™	0-7645-6048-4	\$39.99
Master Microsoft® Office 2000 VISUALLY™	0-7645-6050-6	\$39.99
Master Microsoft® Word 2000 VISUALLY™	0-7645-6046-8	\$39.99
Master Office 97 VISUALLY™	0-7645-6036-0	\$39.99
Master Photoshop® 5.5 VISUALLY™	0-7645-6045-X	\$39.99
Master Red Hat® Linux® VISUALLY™	0-7645-3436-X	\$39.99
Master VISUALLY™ Dreamweaver® 4 and Flash™ 5	0-7645-0855-5	\$39.99
Master VISUALLY™ FrontPage® 2002	0-7645-3580-3	\$39.99
Master VISUALLY™ HTML 4 & XHTML™ 1	0-7645-3454-8	\$39.99
Master VISUALLY™ Microsoft® Windows® Me Millennium Edition	0-7645-3496-3	\$39.99
Master VISUALLY™ Office XP	0-7645-3599-4	\$39.99
Master VISUALLY™ Photoshop® 6	0-7645-3541-2	\$39.99
Master VISUALLY™ Windows® 2000 Server	0-7645-3426-2	\$39.99
Master Windows® 95 VISUALLY™	0-7645-6024-7	\$39.99
Master Windows® 98 VISUALLY™	0-7645-6034-4	\$39.99
Master Windows® 2000 Professional VISUALLY™	0-7645-3421-1	\$39.99

**The Visual™**  
series is available  
wherever books are  
sold, or call  
**1-800-762-2974.**  
Outside the US, call  
**317-572-3993**

# O R D E R F O R M

### TRADE & INDIVIDUAL ORDERS

Phone: **(800) 762-2974**  
 or **(317) 572-3993**  
 (8 a.m.–6 p.m., CST, weekdays)  
 FAX : **(800) 550-2747**  
 or **(317) 572-4002**

### EDUCATIONAL ORDERS & DISCOUNTS

Phone: **(800) 434-2086**  
 (8:30 a.m.–5:00 p.m., CST, weekdays)  
 FAX : **(317) 572-4005**

### CORPORATE ORDERS FOR VISUAL™ SERIES

Phone: **(800) 469-6616**  
 (8 a.m.–5 p.m., EST, weekdays)  
 FAX : **(905) 890-9434**

Qty	ISBN	Title	Price	Total

Shipping & Handling Charges				
	Description	First book	Each add'l. book	Total
<i>Domestic</i>	Normal	\$4.50	\$1.50	\$
	Two Day Air	\$8.50	\$2.50	\$
	Overnight	\$18.00	\$3.00	\$
<i>International</i>	Surface	\$8.00	\$8.00	\$
	Airmail	\$16.00	\$16.00	\$
	DHL Air	\$17.00	\$17.00	\$

Subtotal \_\_\_\_\_

CA residents add  
 applicable sales tax \_\_\_\_\_

IN, MA and MD  
 residents add  
 5% sales tax \_\_\_\_\_

IL residents add  
 6.25% sales tax \_\_\_\_\_

RI residents add  
 7% sales tax \_\_\_\_\_

TX residents add  
 8.25% sales tax \_\_\_\_\_

Shipping \_\_\_\_\_

**Total** \_\_\_\_\_

**Ship to:**

Name \_\_\_\_\_

Address \_\_\_\_\_

Company \_\_\_\_\_

City/State/Zip \_\_\_\_\_

Daytime Phone \_\_\_\_\_

**Payment:**       Check to Hungry Minds (US Funds Only)

Visa    MasterCard    American Express

Card # \_\_\_\_\_ Exp. \_\_\_\_\_ Signature \_\_\_\_\_

