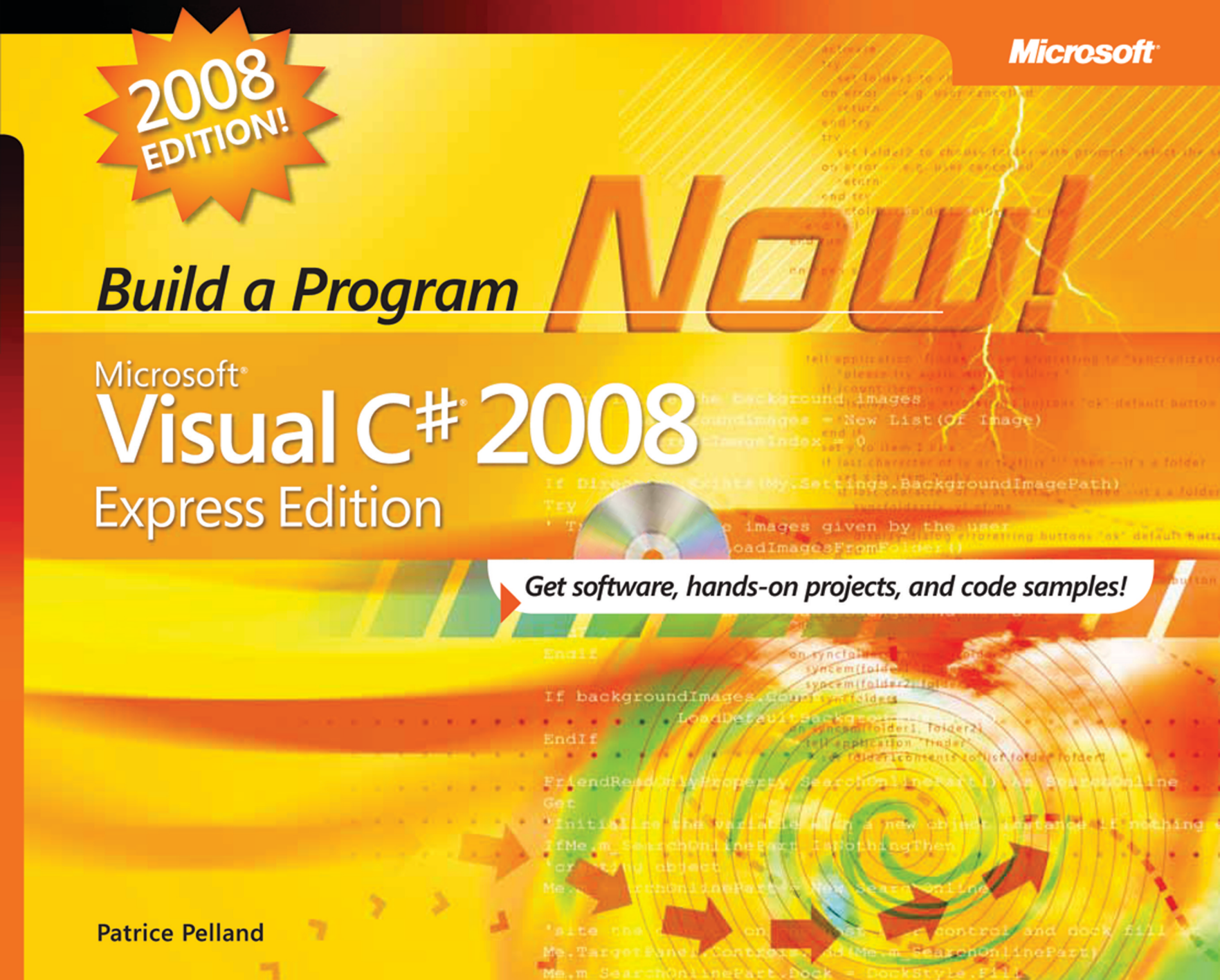
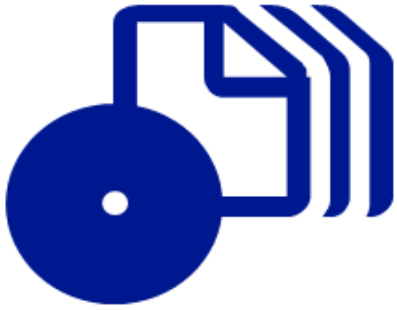

2008 EDITION!
Microsoft
Build a Program Now!
Microsoft® Visual C#® 2008
Express Edition
Get software, hands-on projects, and code samples!
Patrice Pelland

How to access your CD files



The print edition of this book includes a CD. To access the CD files, go to http://aka.ms/625426/files, and look for the Downloads tab.

Note: Use a desktop web browser, as files may not be accessible from all ereader devices.

Questions? Please contact: mspinput@microsoft.com

Microsoft Press

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Control Number: 2008920573

Printed and bound in the United States of America.

1 2 3 4 5 6 7 8 9 QWT 3 2 1 0 9 8

Distributed in Canada by H.B. Fenn and Company Ltd.

A CIP catalogue record for this book is available from the British Library.

Microsoft Press books are available through booksellers and distributors worldwide. For further information about international editions, contact your local Microsoft Corporation office or contact Microsoft Press International directly at fax (425) 936-7329. Visit our Web site at www.microsoft.com/mspress. Send comments to mspinput@microsoft.com.

Acquisitions Editor: Ben Ryan
Developmental Editor: Sandra Haynes
Project Manager: John Pierce

Editorial Production: Happenstance Type O Rama
Technical Reviewer: Dan Maharry

Body Part No. X14-55517

Contents

Introduction

Microsoft Visual C# 2008 Express Edition (and the other Visual Studio 2008 Express Edition products) is, in my opinion, one of the best and most intelligent ideas to come from the Developer Division at Microsoft. I'm applauding and cheering for the people who had this brilliant idea because I believe there is a real need and demand for a world-class, powerful product for hobbyist programmers, students, and professional developers. And Visual C# 2008 Express Edition provides all of that and more.

Visual C# 2008 Express Edition is a fully functional subset of Visual Studio 2008, suitable for creating and maintaining Windows applications and libraries. It's not a timed-bomb edition, a demo, or a feature-limited version—no, it's a key Microsoft initiative to reach more people and give them the ability to have fun while creating cool software.

Who Is This Book For?

This book is for everybody: students, hobbyist programmers, and people who always thought programming was a tough task. It's for people who have ideas like "I wish I could build a tool to store all my recipes and then print them and send them to my friends," "I wish I could build this cool card game that I have never found anywhere else," "I wish I could build this cool software to store my DVD and CD collection," "I wish I could build this software to help me work with matrices and plot graphics for my math class," and many more projects that you can imagine!

This book is for people who have ideas but don't know how to bring them to reality. And it's a good introduction to the art and science of developing software.

How This Book Is Organized

This book consists of nine chapters, each covering a particular feature or technology about Visual C# 2008 Express Edition. Most chapters build on previous chapters, so you should plan on reading the material sequentially.

Conventions and Features in This Book

This book presents information using conventions designed to make the information readable and easy to follow. Before you start the book, read the following list, which explains conventions you'll see throughout the book and points out helpful features in the book that you might want to use:

- Each exercise is a series of tasks. Each task is presented as a series of numbered steps (step 1, 2, and so on). Each exercise is preceded by a procedural heading that lets you know what you will accomplish in the exercise.
- Boxes labeled *TIP*, *NOTE*, *MORE INFO*, and so on provide additional information or alternative methods for completing a step successfully.

- Boxes labeled *CAUTION* alert you to information you need to verify before continuing.
- Text that you type appears in bold.
- Menu commands, dialog box titles, and other user interface elements appear with each word capitalized, such as in "click Save As."
- A plus sign (+) between two key names means that you must press those keys at the same time. For example, "Press Alt+Tab" means that you hold down the Alt key while you press the Tab key.
- Code listings appear in a monospaced font in this book.
- Sidebars throughout the book provide more in-depth information about the content. The sidebars might contain background information, design tips, or features related to the information being discussed.
- Each chapter ends with an "In Summary..." section that briefly reviews what you learned in the current chapter and previews what the next chapter will present.

System Requirements

You'll need the following hardware and software to complete the exercises in this book:

- Windows Vista, Microsoft Windows XP with Service Pack 2, or Microsoft Windows Server 2003 with Service Pack 2
- Visual C# 2008 Express Edition
- 1 GHz 32-bit (x86) processor
- 1 GB MB RAM (512 MB minimum)
- 40 GB hard drive with at least 15 GB of available space
- Support for Super VGA graphics (for support for DirectX 9 graphics, see the recommended requirements at *www.microsoft.com/windows/products/windowsvista/editions/systemrequirements.mspx*)
- CD-ROM or DVD-ROM drive
- Microsoft mouse or compatible pointing device

You'll also need administrator access to your computer to configure SQL Server 2005 Express Edition.

NOTE

The companion DVD contains the Visual C# 2008 Express Edition software needed to complete the exercises in this book. The DVD also includes the other Visual Studio 2008 Express Editions—for Visual Basic, Visual C++, and Web development. You can install any of the Express Edition products included on the DVD. See Chapter 2, "Installing Visual C# 2008 Express Edition" for detailed installation instructions.

Code Samples

You can download the code samples for the examples in this book from the book's companion content page at the following address: *http://www.microsoft.com/mspress/companion/9780735625426*.

You'll use the code samples and starter solutions as you perform the exercises in the book. By using the code samples, you won't waste time creating files that aren't relevant to the exercise. The files and step-by-step instructions in the lessons also let you learn by doing, which is an easy and effective way to acquire and remember new skills. You'll also find the complete solutions if you want to verify your work or if you simply want to look at it.

Installing the Code Samples

Follow these steps to install the code samples on your computer.

1. Download the code samples from *http://www.microsoft.com/mspress/companion/9780735625426*.
2. After you download the code samples file, run the installer.
3. Follow the instructions that appear.

   The code samples are installed in the Documents\Microsoft Press\VCS 2008 Express folder on your computer.

Using the Code Samples

Each chapter in this book explains when and how to use any code samples for that chapter. When it's time to use a code sample, the book will list the instructions for how to open the files. The chapters are built around scenarios that simulate real programming projects so you can easily apply the skills you learn to your own work.

For those of you who like to know all the details, a list of the code sample projects appears in the following table. Almost all projects have solutions available for the practice exercises. The solutions for each project are included in the folder for each chapter and are labeled "Complete."

Project	Description
Chapters 1 and 2	No sample projects.
Chapter 3	
MyFirstConsoleApplication	Application that takes two numbers, adds them together, and then displays the sum in a console window.
MyFirstWindowsApplication	Same application as MyFirstConsoleApplication, but this one displays the result in a message box.
Chapter 4	Simple Web browser application that enables the user to browse on the Internet.
MyOwnBrowser	
Chapter 5	Application that teaches you to use the most important features in Visual C# 2008 Express Edition.
TestProject	
Chapter 6	This is the same application you developed in Chapter 4, enhanced with additional features. You'll add menus, toolbars, status and progress bars, and a navigation window with autocomplete. You'll also build a simple browser using Windows Presentation Foundation (WPF).
MyOwnBrowser	

Project	Description
Chapter 7 Debugger	An application full of problems to help you learn how to debug an application by using features of Visual C# 2008 Express Edition.
Chapter 8 CarTracker	An application enabling the user to track car ads from the Internet using a SQL Server 2005 Express database to store the information. You'll also be introduced to Language Integrated Query (LINQ).
Chapter 9 Weather Tracker	An application that runs in the system tray and has a nice user interface to display weather data collected by your application from the MSN Weather service. You'll also create a deployment package for the distribution of your application.

Uninstalling the Code Samples

Follow these steps to remove the code samples from your computer.

ON WINDOWS VISTA

1. In Control Panel, click Programs.
2. Under Programs and Features, click Uninstall a Program.
3. In the list of programs, select Microsoft Visual C# 2008 Express Edition: Build a Program Now!, and then click Uninstall.
4. Follow the instructions on the screen to remove the code samples.

ON WINDOWS XP

1. In Control Panel, open Add or Remove Programs.
2. From the Currently Installed Programs list, select Microsoft Visual C# 2008 Express Edition: Build a Program Now!, and click Remove.
3. Follow the instructions on the screen to remove the code samples.

Prerelease Software

This book was reviewed and tested against the November 2007 release candidate for Visual Studio 2008. This book is expected to be fully compatible with the final release of Visual Studio 2008. If there are any changes or corrections for this book, they'll be collected and added to a Microsoft Knowledge Base article. See the "Support for This Book" section later in this introduction for more information.

Technology Updates

As technologies related to this book are updated, links to additional information will be added to the Microsoft Press

Technology Updates Web page (*http://www.microsoft.com/mspress/updates/*). Visit this page periodically for updates on Visual Studio 2008 and other technologies.

Support for This Book

Every effort has been made to ensure the accuracy of this book and the companion content. As corrections or changes are collected, they'll be added to a Microsoft Knowledge Base article. To view the list of known corrections for this book, visit *http://support.microsoft.com/* and in the Search box, enter the book title.

Microsoft Press provides support for books and companion content at *http://www.microsoft.com/learning/support/books/*.

Questions and Comments

If you have comments, questions, or ideas regarding the book or the companion content or have questions that are not answered by visiting the sites listed earlier, please send them to Microsoft Press via e-mail to *mspinput@microsoft.com*.

Or you can send them via postal mail to the following address:

Microsoft Press
Attn: Visual C# 2008 Express Edition: Build a Program Now! Editor
One Microsoft Way
Redmond, WA 98052-6399

Please note that Microsoft offers no software product support through these addresses.

About the Author

Patrice Pelland is a development manager at Microsoft working in the Online Services Group. He has a passion for Web 2.0 technologies, Silverlight, WCF, and ASP.NET. For the past four years, he has been working, teaching, evangelizing, and talking about these technologies to everyone.

For the past 14 years, he has been working in software development in various roles: developer, project lead, manager and mentor, and software engineer in QA organizations. He has vast experience spanning multiple technologies and fields, including Web development, developer tools, fiber optics telecommunication, aviation, and coffee and dairy companies. He also spent three years teaching computer science and software development at a college in Canada.

When not developing great tools for developers and helping customers throughout the world, he enjoys spending time with his family and friends, playing games on Xbox 360 and his PC, reading books, reading about cars, playing hockey, watching NHL hockey and NFL football, and having great dinners with good food and fine drinks with friends and family. He resides with his family in Sammamish, Washington.

Dedication

This book is dedicated to my family. My wife, Hélène, is my strength; because of her love and her respect, I am a better human being. She's beautiful—my idol, my inspiration, my sunshine, my best friend, my love, and an awesome mother! *Mon amour*, thanks for being who you are and for being there for me. I love you! Thanks to her for letting me repeat this crazy adventure of writing a book.

Thanks

First of all, thanks to my parents. Mom and Dad, you gave me all the chances to be what I am in life and you gave me the values to be the man I am. Thanks, and I love you!

A book is a huge adventure in somebody's life (imagine two ), and it would not be possible without the help of many people. I've always read the "thank you" sections in other people's books, and I was always amazed at how many people are needed to make a book what it is. Now I really understand why!

Although writing a book is tough—real tough—it's really satisfying at the same time. During the writing process, you sometimes have doubts, and I had my share of them, especially those nights at 3 A.M. when all other souls in the house were asleep, even my dog; when I was in front of my laptop with an exception and a white page in Microsoft Word. I can't remember how many times I said to my friends, "No, I won't be able to be there. I need to work on my book." But it's an awesome experience to write a book; everybody who has the chance should take the challenge!

That said, I first need to thank my lovely family for letting me do this to them again. My kids (Laura, 13, and Antoine, 11) and my wife, Hélène, were so great and *patient*. This time they said, "You're writing another book! Oh, no...we'll see you after Thanksgiving." But at the same time, they were respecting the space I needed and the time alone! You guys are great, and I love you!

I have to thank all the people at Microsoft Learning and the publishing team. I would especially like to thank Ben Ryan for offering me the chance to work with him again; Sandra for her constant motivation, help, and suggestions and also for helping me through all the hurdles of writing a book; and all the folks on the publishing team for all their help getting the job done and producing a real, tangible product. You guys have my respect for working day in, day out in the crazy world of publishing.

I would also like to thank all the people in the Visual Basic, C#, Windows Forms, MSDN, and setup teams who helped me by answering all my questions in a dynamic and constantly changing product life cycle. I would like to thank more specifically Dan Fernandez, Joe Binder, Brian Keller, Brian Johnson, Hong Gao, Jay Roxe, Kavitha Radhakrishnan, Kent Sharkey, Lisa Feigenbaum, Shamez Rajan, Steve Lasker, Aaron Stebner, and Habib Heydarian.

Thanks also to my colleagues at MSN for always giving me good words of encouragement and to my friends Pascal, Simon, Nicolas, John, and Patrice for reviewing the samples and some chapters.

Thanks to my good friends here in the Puget Sound area for the kind words of encouragement and to my family and friends in Canada for understanding why I'm not calling or giving any news. Sorry, Mom and Dad!

Thanks to everybody I might have forgotten!

Patrice Pelland
November 2007
Sammamish, WA

Chapter 1

Introducing Microsoft Visual C# 2008 Express Edition

Maybe you've decided to try programming and find yourself with this book. If that's the case, you've come to the right place. This book is all about introducing you to the art, science, and joys of creating software for Microsoft Windows—yes, the same Microsoft Windows you probably use every day. Throughout the book, I'll show you how to build applications that are similar to many of the applications you use on a regular basis, such as your Internet browser, your word processor, your e-mail software, and your personal finance application. You're probably wondering how you could possibly do this with no programming experience. Don't worry. By the time you finish this book, you'll be a believer. We'll have a blast, and because you'll actually be building applications as you follow along with each exercise, you'll see for yourself just how easy it can be.

What Is .NET?

What is this .NET thing everybody is talking about? Maybe you've seen the term somewhere online or come across it in the jobs section of your Sunday newspaper. A good analogy is that .NET—also called the .NET Framework—is to a software developer what tools and manuals are to an auto mechanic.

Here is a formal definition of the .NET Framework:

> *The .NET Framework is a platform with which you can develop software applications and libraries called* managed applications*; it provides you with the compiler and tools you need to build, debug, and execute managed applications.*

For our purposes, you could say that .NET is the platform that gives you everything you need to develop and run managed applications that run on Windows.

We say that applications are *managed* because their execution is managed by the .NET Framework. In fact, the .NET Framework manages the execution by providing a controlled runtime environment that offers a wide variety of services, such as loading your applications, managing memory, and monitoring and maintaining security and integrity while the application runs. Before .NET (and Java), applications were unmanaged because they were not executed by a controlled runtime environment. No other component of the operating system provides the services .NET offers. The applications had to manage their own services, which sometimes led to erroneous code, security holes, and data corruption. Because of these problems, applications were tough to maintain and debug.

NOTE

Throughout this book, I'll use the terms *framework* and *.NET Framework* synonymously.

The .NET Framework provides you with a wide variety of tools, such as compilers, debuggers, programming languages, an execution engine (named the Common Language Runtime [CLR]), developer tools, and a large number of predefined "building block" libraries. These libraries are named Framework Class Libraries (FCLs). You can think of each .NET component as a building block in a house and each version of .NET as an insulation layer in the walls of the house. Figure 1-1 illustrates how many versions of .NET are out in the market, what components have been added, in which version they belong.

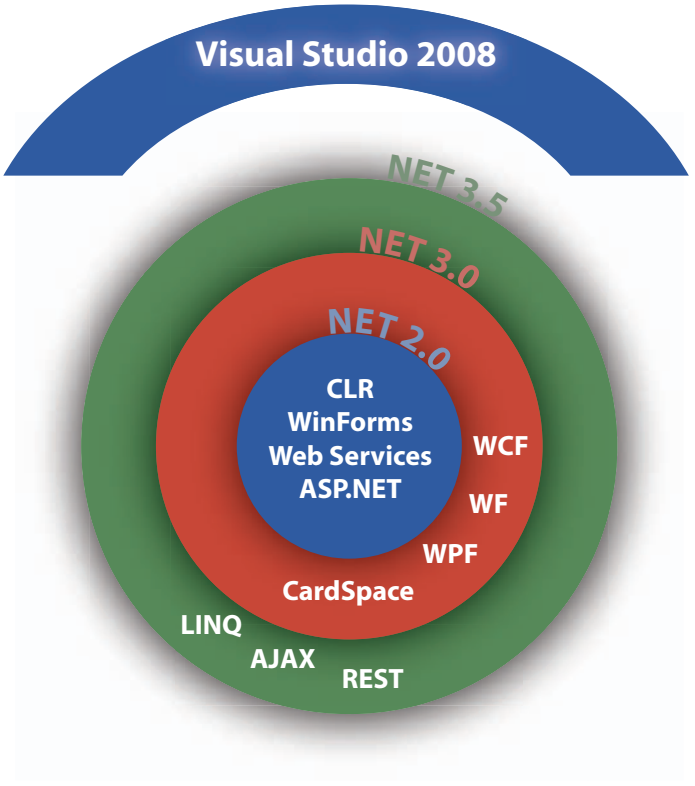


Figure 1-1
Additive versions of the .NET Framework

NOTE

What do the other abbreviations and names in Figure 1-1 mean? WF is the Windows Workflow Foundation, another building block that developers can use to help automate business processes through programs. CardSpace is a technology related to managing online identities—something like using a credit card. AJAX (Asynchronous JavaScript And XML) is used to develop Web applications, and REST (Representational State Transfer) is a programming architecture used for transferring data on the Web.

Some of these building blocks ship with the Windows Vista operating system. Two popular ones are Windows Presentation Foundation (WPF) and Windows Communication Foundation (WCF). WPF is a library that helps you build richer user interfaces and Windows Vista–like applications for Windows. WCF, as its name implies, is a library that helps two applications talk to each other using messages. To understand the relationship between .NET 3.0 and .NET 3.5, remember that .NET 3.0 comes with Windows Vista and .NET 3.5 comes with Visual Studio 2008. Language Integrated Query (LINQ), which simplifies writing code that manipulates data from various data sources (SQL Server databases, XML files, and so on), is one of the features in .NET 3.5 that we'll cover in this book.

IMPORTANT

It's not necessary to have Visual Studio to develop .NET applications, but using it offers many advantages, as you'll see in this book.

IMPORTANT

The CLR hasn't changed in Windows Vista and Visual Studio 2008; the CLR that is running on all operating systems is .NET 2.0.

I won't put you to sleep with all the definitions for each building block. We're going to use or talk about most of them in our projects in this book, and I'll introduce the blocks when appropriate. Just consider Figure 1-1, and return to it when you need to do so.

Two notes about this image are worth mentioning.

First, look at the blue component on top of the concentric circles. Microsoft Visual Studio 2008 is not part of the .NET Framework, but it touches the .NET Framework at all levels. With Visual Studio 2008, you can develop applications that take advantage of all the components of the .NET Framework.

Second, notice that the CLR, among other components, is at the center of the circles. The CLR is a crucial part of the foundation because it's the engine that loads and manages the execution of source code.

What Is C#?

C# is one of the programming languages that targets the .NET Framework. Like any spoken or written language, C# has syntax rules and a series of valid words you can use to create your applications. C# is a popular choice for beginners because some people find the syntax simpler than the syntax of many other programming languages.

Is C# an Object-Oriented Programming Language?

C# is a fully fledged object-oriented programming language. Let's talk about what this means.

Object-oriented programming (OOP) is a programming style (or programming paradigm). There are other programming paradigms, such as functional or procedural programming. Languages such as C, Fortran, and Pascal all use functional or procedural programming paradigms. These paradigms focus more on actions, while OOP focuses more on the data itself.

Applications that use the OOP paradigm are developed using OOP languages (OOPLs). The first OOPLs were introduced in the late 1960s, but they really became popular in the late

1970s. They are widely used today because most people agree that they're easy to learn, use, debug, and maintain. For instance, OOPLs easily represent real-world objects. C# is an OOPL, as are Visual Basic .NET, C++, Java, Smalltalk, Lisp, and others.

MORE INFO

With C++ you can develop procedural applications, pure object-oriented applications, or a mix of both.

Programmers use OOP to write programs that contain representations of real-world objects that are known as *classes* or *types*. You can think of an OOP program as a collection of objects interacting with each other. Using OOP, a programmer defines new types to represent real-world objects, such as a plane, a person, a customer, a dog, or a car. Those types or classes have constructors that developers use to create objects or instances. An *object* is a unit that represents one instance of a real-world object. It's a self-contained unit because it includes all the data and functionality associated with that object. This means each object created in an application contains all the information that characterizes it (*data members* or *fields*) and all the actions (*methods*) that can access or modify that information.

Here is a simple example in C# that defines a *Person* class:

```
1  using System;
2
3  public class Person
4  {
5      //Data members
6      private string Name;
7      private string Address;
8      private string City;
9      private string State;
10     private string ZIP;
11     private string Country;
12
13      // Methods
14      public virtual void Display()
15      {
16          Console.WriteLine(Name);
17          Console.WriteLine(Address);
18          Console.WriteLine(City);
19          Console.WriteLine(State);
20          Console.WriteLine(ZIP);
21          Console.WriteLine(Country);
22      }
23 }
```

MORE INFO

In the example of the *Person* class, you would need to implement properties to access or modify the *Private* fields from outside the class. These types of fields hide data in your class.

This class includes private data members and a *Display* method to print the object's content to the console. The *virtual* keyword used in the definition of the *Display* method means that a class derived from this class will be able to write its own implementation of the *Display* method.

Let's use a different example to go over these concepts some more. My dog, Chopin, is an instance of the class *Dog*, and the class *Dog* is a subclass of the *Animal* class. Because Chopin is a dog, he has some behaviors and data that are proper for a dog. But because a dog is also an animal, Chopin also inherits some data and behaviors from the *Animal* class.

This means that the instance *Chopin* of the *Dog* class has data members that characterize him and methods that I can call on that little furry ball. For example, here is the instance information for the *Chopin* object:

Data

- **Breed** He's a Maltese.
- **Gender** He's male.
- **Weight** His weight is 5.5 pounds (2.5 kilograms).
- **Color** He's white.
- **Name** His name is Chopin Chabispel.
- **Age** He's 1.5 years old.

Actions

- He speaks (barks).
- He eats.
- He moves.
- He sleeps.

All these data items (breed, gender, weight, color, name, and age) and actions (speak, eat, move, and sleep) characterize him, but they can also characterize any other dog, such as my neighbor's dog, Molly. And if you think about it, those items can characterize any animal. This means that the class *Dog* inherits data members and methods from the class *Animal*.

Let's say you want to develop an application for a veterinary clinic. To cover the cats who come to your clinic, all you must do is create a *Cat* class that also inherits from the class *Animal*. Then each class (*Cat* or *Dog*) could override functionality from the *Animal* class as needed. For instance, for the *Cat* class, the *Speak* method would be *meow* instead of *bark*. This means that the *Speak* methods for *Cat* and *Dog* are specializations of the regular animal *Speak* method.

Let's look at the *Person* class example again. This time, I'll also show an *Employee* class that derives from the *Person* class. The *Employee* class derives from the *Person* class by using a colon (:) followed by the *Person* element. The keyword *override* changes the implementation of the *Display* method.

```
52  public class Employee : Person
53
54  {
55      public int Level;
56      public int Salary;
57
58      public override void Display()
59      {
60          Console.WriteLine(Name + " is at level " + Level.ToString() + "
              and has a salary of : " + Salary.ToString() + "$");
61          Console.WriteLine("His address is:");
62          Console.WriteLine(Address);
63          Console.WriteLine(City + "," + State + " " + ZIP);
64          Console.WriteLine(Country);
65      }
66  }
```

TIP

In this book, you'll notice that some code listings include line numbers. If a line does not include a number, it indicates that the code is a continuation from the previous line. Some code lines can get rather long and must be wrapped to be displayed on the printed page. If you need to type in the code in Visual C#, be sure to put continued lines on a single line.

In this case, the *Employee* class inherits from the *Person* class and therefore gets all the data fields from that base class. The *Employee* class doesn't have to redefine any of the fields in its definition because it gets them automatically from *Person*. So, for the *Employee* class, you must specify only what is different from an instance of the *Person* class. For example, an instance of the *Employee* class would have *Level* and *Salary*, whereas none of the instances of the *Person* class would have these. Plus, the *Display* method for *Employee* could thus add *Level* and *Salary* information to the displayed message when it is called.

SEE ALSO

In .NET, all classes ultimately derive from the *Object* class, even when it is not specified.

This was just a brief introduction to OOP and some of its concepts. C# supports all of these concepts and many more. Throughout this book you'll see more OOP concepts, and when you do, I'll highlight them in a "reader aid" information box, as shown in the left margin.

Here's the complete listing used in this section with the addition of the *Customer* class:

```
1  using System;
2
3  public class Person
4  {
5      //Data members
6      public string Name;
7      public string Address;
8      public string City;
9      public string State;
10     public string ZIP;
11     public string Country;
12
13      // Methods
14      public virtual void Display()
15      {
16          Console.WriteLine(Name);
17          Console.WriteLine(Address);
18          Console.WriteLine(City);
19          Console.WriteLine(State);
20          Console.WriteLine(ZIP);
21          Console.WriteLine(Country);
22      }
23 }
24
25 public class Customer : Person
26
27 {
28      public int ID;
29      public bool IsPartner;
30
31      public override void Display()
32      {
33          string partnerMessage;
```

```
34
35          if (IsPartner)
36          {
37              partnerMessage = " is a partner";
38          }
39          else
40          {
41              partnerMessage = " is not a partner";
42          }
43
44          Console.WriteLine("Customer ID: " + ID.ToString());
45          Console.WriteLine(Name + partnerMessage);
46          Console.WriteLine(Address);
47          Console.WriteLine(City + "," + State + " " + ZIP);
48          Console.WriteLine(Country);
49      }
50  }
51
52  public class Employee : Person
53
54  {
55      public int Level;
56      public int Salary;
57
58      public override void Display()
59      {
60          Console.WriteLine(Name + " is at level " + Level.ToString() +
                " and has a salary of : " + Salary.ToString() + "$");
61          Console.WriteLine("His address is:");
62          Console.WriteLine(Address);
63          Console.WriteLine(City + "," + State + " " + ZIP);
64          Console.WriteLine(Country);
65      }
66  }
```

This is a simple case, but it illustrates some of the basic concepts of OOP.

Visual C# 2008 Express Edition is the tool we will use throughout this book to develop applications that run on Windows.

The Express editions of Visual Studio 2008 were designed to focus on productivity. As with their high-end version counterparts, the Express editions of Visual Studio 2008 are also what we call *rapid application development* (RAD) tools because their philosophy is geared toward productivity. The Express editions of Visual Studio are easy to use, easy to learn, and streamlined because although they contain mostly the same components, they lack the full breadth of features found in the higher-end versions of Visual Studio. Most features and components in the Express editions were simplified to make the learning curve less steep and to fit the needs of the nonprofessional developer.

The Visual Studio 2008 Express Edition family is designed with beginner programmers in mind—people like you who are curious about programming and who are looking for an easy way to build Windows applications while learning how to program. Visual C# 2008 Express Edition is the ideal tool to use to rapidly develop applications for topics you really love or for hobbies you enjoy. You can also use it to help ease your day-to-day job or school tasks. Most important, you can have fun with the tool while you're learning to program.

What Kinds of Applications Can You Build with Visual C# 2008 Express Edition?

With this version of Visual Studio 2008, you'll be able to create the following types of applications:

- **Windows applications** These are the applications that have a graphical interface with buttons, windows, menus, toolbars, and so on, as in Microsoft Word or Windows Internet Explorer. With this book you'll learn how to take advantage of WPF, which lets you build applications that create a rich user experience while exploiting all the power of your computer. You can also build applications that look like Windows Vista–based applications.

- **Console applications** These are the applications that have no graphical interface and that simply use text to communicate with the user. (Typically, these applications run in a command window or DOS window.)
- **Reusable components or class libraries** These are groups of tools created to help build other applications.

What you won't be able to build are Web sites and Web services. To create any type of Web application, you will need to get Microsoft Visual Web Developer 2008 Express Edition.

NOTE

We will look into the details of what types of applications fall into these categories in Chapter 3, "Creating Your First Applications."

What Are the Key Features You Need to Know About?

The following list, although not complete, describes the essential features of Visual C# 2008 Express Edition. At this point, don't worry if you don't understand every feature listed. I'm presenting the features in the list because you'll come across all of them in some way in the fun sample applications you will be creating as you read this book.

Most of the features listed here emphasize the RAD philosophy. Although the idea is to give you an overview of the interesting features that can make your life easier, the names of the features alone are not sufficient to understand what they mean. I've included a brief description giving you the essentials and explaining how they will help you develop applications.

- **Built-in Starter Kits** The Starter Kits are fully developed applications with best practices and examples to follow. These applications will give you another example on which to base your learning. They will be a good complement to what we are doing with this book. You can find them at *http://msdn2.microsoft.com/en-us/vcsharp/aa336742.aspx*, and you can find a few others at *http://www.microsoft.com/express/vcsharp/Default.aspx*.
- **Beginner's targeted documentation and tutorials** These are a fast and easy way to get information. They also provide samples.
- **IntelliSense** This feature provides real-time syntax suggestions and even finishes your typing for you. In Visual Studio 2008, IntelliSense, as you will see, is everywhere, and it provides a more complete and contextual set of suggestions than in earlier versions of Visual Studio.

- **Code snippets** Snippets provide code for various programming tasks to help you complete many common tasks automatically. In addition, code snippets show the recommended way of performing a task. They are directly integrated into the development environment, and they are extensible; that is, anybody can extend the existing snippets or provide new ones. Over time Microsoft will continue to supply new code snippets, and members of online communities will contribute their snippets as well. Code snippet extensibility is a really nice feature that helps people share useful features in online communities.
- **Data-enabled applications** With these applications you can connect to Microsoft SQL Server 2005 Express Edition and add databases and code to access the data in your applications. In addition, a new editor has been added to help you develop applications that use data. As mentioned earlier, LINQ is one of the big new features of .NET 3.5 included with Visual Studio 2008, and you will see how to use it in Chapter 8, "Managing the Data."
- **Windows Forms Designer and WPF Designer** With these new tools, you can easily design your Windows application using either Windows Forms or WPF, including features such as snap lines, which make sure your controls are aligned in your form.
- **XAML editor** The XAML editor lets you edit Extensible Application Markup Language (XAML), which was introduced with .NET 3.0. This new markup language is used extensively in WPF and Windows Workflow (WF) to describe user interface elements in WPF and process logic in WF. (WF is beyond the scope of this book.)
- **XML Web services** Visual C# 2008 provides easy-to-use tools and wizards that help you connect to published XML-based Web services and utilize their functionalities.
- **New Windows Forms controls** These comprise an impressive list of controls—a greater selection than in any previous version of Visual C#. They will help you create user interfaces that have a professional look and feel.
- **Smart Tags** Most Windows Forms controls that come with the product include Smart Tags. As in many applications of the 2007 Microsoft Office system, a Smart Tag is represented by a little black triangle or an icon and a little black triangle attached to a control. A Smart Tag gives you access to the most common actions you can perform on a control.

MORE INFO

XAML is also used in Silverlight for Web applications, but this is beyond the scope of this book.

- **Refactoring** The Visual C# 2008 IDE includes robust and powerful refactoring support. By *refactoring*, developers can automate many common tasks related to restructuring code. *Restructuring* code is when you want to change some of your source code elements and apply that change to all files and occurrences of that element. For instance, you'll be able to rename variables throughout a project, promote local variables to parameters, extract methods, and do much more. The Visual C# 2008 IDE gives you a nice preview feature so that you can see the changes before you make them. You'll be able to access the refactoring features either by opening a context-sensitive menu while editing your source code or by using a Smart Tag. For more information on refactoring, please visit *http://www.refactoring.com/*, and for more examples on how to implement these features in the Visual C# 2008 IDE, visit *http://msdn2.microsoft.com/en-us/library/719exd8s(VS.90).aspx*.
- **Organizing Using feature** Related to the refactoring feature, the Organizing Using feature lets you clean your source code by removing unused *using* statements or sorting them without changing the behavior of the source code. Over time you might add features requiring the addition of new assemblies, which usually means new *using* statements. Over the course of your application's development, you might change your mind, and then your code might be bloated with unwanted artifacts such as *using* statements. With the Organizing Using feature, you can delete the ones you are not using in your code. You can also remove and sort them in one pass.
- **ClickOnce deployment** With this feature, you can easily publish your applications to the Internet, on a local area network (LAN), on a network share, or on a CD or DVD. It also simplifies publishing updates. With this edition of Visual Studio, you can now handle with the ClickOnce wizard the Windows Vista User Account Control (UAC) so that your application runs in the lowest user security context it needs. Usually you want to aim your software development on Windows Vista for regular users. This has the effect of reassuring the user that your application won't perform unsafe operations without their knowledge.
- **Edit and Continue** While you are debugging your application, the Edit and Continue feature lets you modify the code, move back and forth in the debugger, re-execute code, add functionality, or fix bugs on the fly without stopping program execution.

- **Debugger visualizers** While you are debugging your application, visualizers give you an easy way to get readable representations of your application data. They give you a human-readable representation of the stored data, even for more complex types found in ADO.NET or XML.
- **Community Access and Start pages** With these features, you can access additional information from online communities and from different sources of online help, including diverse RSS feeds. (RSS can stand for Rich Site Summary or Really Simple Syndication and is a family of XML file formats; it is widely used by the blog community and news Web sites.)
- **Simplified development environment** Everything in the development environment was created so that you can easily access key functionalities, tools, and objects.

As you can see, Visual C# 2008 Express Edition includes many nice features to help new programmers develop applications in a fast and fun way. These features will provide guidance even when you're not necessarily sure what syntax or components to use and will greatly expedite learning the product.

In Summary...

You now know that .NET is a framework composed of compilers, tools, languages, debuggers, and an execution engine. The CLR is that execution engine, and it is responsible for loading and executing managed applications. In essence, .NET is like a house with the CLR as the foundation and all other services built on top of it. You also learned that the CLR didn't change with Windows Vista and Visual Studio 2008, but a lot of new building blocks have been added so you can take advantage of features provided by Windows Vista and make developing applications easier.

In addition, you learned that C# is an object-oriented programming language. You also started to learn what object-oriented programming is and the basics of OOP in Visual C# 2008.

This chapter gave you the opportunity to hear about the most important features of Visual C# 2008 Express Edition. In the next chapter, you'll learn how to install Visual C# 2008 Express Edition.

Chapter 4

Creating Your Own Web Browser in Less Than Five Minutes

Now that you've gotten a little experience creating simple applications in Microsoft Visual C# 2008 Express Edition, you'll build a more complicated application in this chapter and finish it in Chapter 6, "Modifying Your Web Browser." In this chapter, you'll start with the basic framework of the application; in the next two chapters, you'll continue to learn new features and then use them to enhance your project.

Specifically, in this chapter you'll learn how to build your own basic Web browser, and you'll be able to do it in five minutes or less!

What Is a Project?

In the previous chapter, you created a project to hold your source code. I'll now take a moment to explain what a project is and what information it contains. A *project* is a container for all the items in your application, such as forms, source code, and resources. It also stores important configuration data that belongs to the application as a whole: the location of the executable (that is, binaries) on your hard disk, the version information, and many more settings that affect the characteristics of your application. For instance, a project stores programmer-defined application settings that are important for the user experience. Users love to customize their software environment to reflect their comfort level and personal styles, for example. You've probably set up specific user preferences in Windows Internet Explorer, such as your home page address, your home page settings, which toolbars are displayed, whether your toolbars are locked in size, and so forth. A typical use of application settings in a project is to make sure the application can preserve user customizations from one execution to another.

In the next chapter, you'll learn about some of the most important settings stored in the project configuration file and how to use them in your application. In the final chapter of this book, you will use programmatic techniques to preserve the user's settings and customizations.

The name you choose when you create your application becomes your project's name. It also becomes the default folder name on your hard disk where your application is stored when you save it, and this name becomes the default namespace of your application. A *namespace* is used to organize the classes in a program in a single, logical hierarchical structure. It does the same for any other types you might define. The creation of a namespace also helps prevent naming collisions. What is a naming collision? Let's look at an example to illustrate this concept.

Suppose a company called AdventureWorks wrote a new Windows Forms class named *ANewForm*. The company would create a namespace called *AdventureWorks* and put its *ANewForm* class in it to uniquely name the class. The fully qualified name of a class is always

composed of the namespace followed by a dot and then the name of the class or classes. Therefore, AdventureWorks's unique class would be *AdventureWorks.ANewForm*.

Now let's suppose you are creating a new project using Visual Studio and decide to name your project MyLibrary. Visual Studio would then create for you a namespace called *MyLibrary*. Suppose you then define a new class and name it *ANewForm*. You might not be aware that a company called AdventureWorks also called its new class using the same name. Even though AdventureWorks might be performing completely different tasks with its class, a problem could arise because the two classes are named the same.

Now suppose you're trying to use both classes called *ANewForm* in your new application. If you simply use *ANewForm*, the compiler will not be able to determine which *ANewForm* class you want to use—the one from your library or the one from the AdventureWorks library; this is a *naming collision*. By prefixing the class name with the namespace name, you are then telling the compiler exactly which class you want to use (*AdventureWorks.ANewForm* or *MyLibrary.ANewForm*).

What Is the Design Layout?

You will soon create a new design layout in the form designer. In doing so, you'll be creating what the application contains and how its content is presented when the user executes the application.

To accomplish this phase of a project, you typically do not need to type a great deal of code; as explained later in this chapter, Visual Studio takes care of this code for you. You have to worry mostly about how your application looks. When you're done designing all the visual aspects to your liking, your next task usually involves attaching the source code to your visual layout so that your application reacts to and acts upon the user's input.

In this chapter, you will complete the basic layout. You will learn more advanced layout techniques in following chapters. Let's start the Web browser project now.

TO CREATE A SIMPLE WEB BROWSER

1. Start Visual C# 2008 Express Edition by clicking Start, All Programs, Microsoft Visual C# 2008 Express Edition.
2. Create a new Windows Forms application by using any of the techniques shown in the previous chapters; for instance, you can use either the File menu or the New Project icon in the toolbar. Name the new application MyOwnBrowser.
3. On the design surface, you'll see the empty form with a title bar named Form1. Click the title bar once. By default, you don't see the Properties window. To view it, select the View menu, and then select Properties Window (or press F4). Look at the Properties window on the bottom right of the IDE, as shown in Figure 4-1.

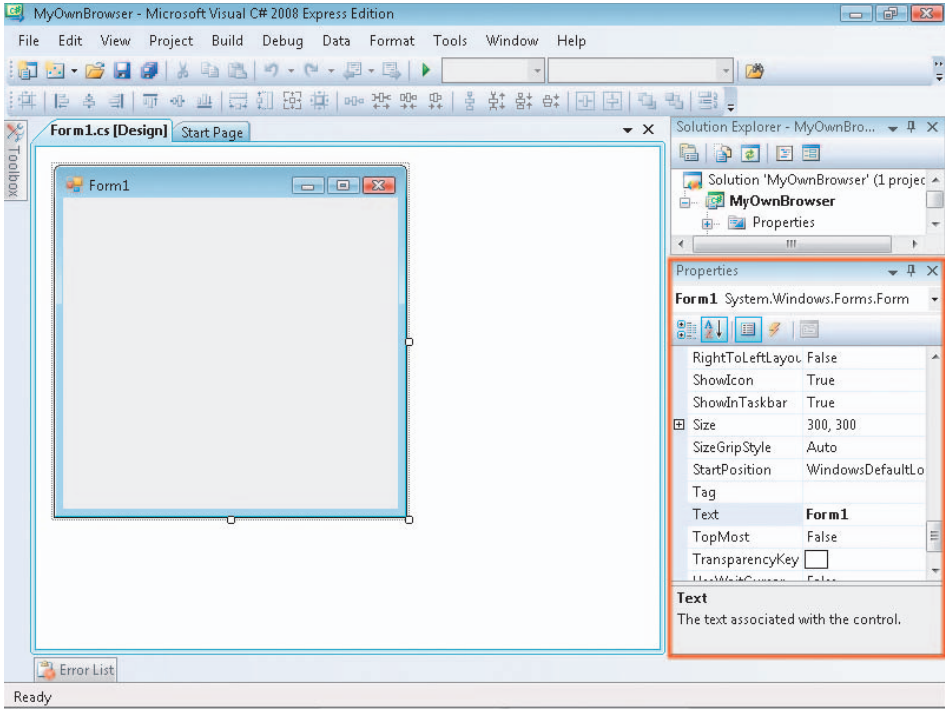


Figure 4-1
Properties window for MyOwnBrowser application Form control

We'll be using most of the properties you see listed here. Right now what is important for you to understand is that most of these properties influence how the control you have selected behaves or what it looks like when you execute your application.

For all the samples in this book, I suggest you sort the Properties window in ascending alphabetical order; it will be much easier to find properties that I reference in the examples. To sort the properties in ascending alphabetical order, click the Alphabetical button (with the AZ icon) in the Properties window toolbar. The other option is to arrange the controls by category, but this might slow you down as you progress through this book.

Whenever you select a property, you'll see a brief description of its usage at the bottom of the Properties window. Refer to Figure 4-1 as an example. In this case, the *Text* property is selected, and at the bottom of the Properties window, you can see a succinct message describing the function of the *Text* property.

As mentioned in Chapter 3, "Creating Your First Applications," my best advice for learning this software is to try, try, and try again. Visual C# 2008 Express Edition comes with a variety of tools and therefore many possibilities. You will learn to use most of these tools by performing the exercises in this book, but it's impossible to learn all the variations and possibilities if you don't do some exploring on your own. With that in mind, to understand the effect of changing a particular property, try all the possible values. Each time you modify a property, build and verify the execution. However, don't make more than one change at a time. If you do, it will be difficult for you to know which one of your changes actually triggered a visual modification. By exploring possibilities one at a time, you'll be able to see the effect of your changes immediately.

IMPORTANT

Some properties have a plus (+) sign beside them, which means they're tree view properties. Whenever you click the +, you'll expand this property to display the property's attributes, which you will then be able to set. Whenever you are asked to enter values for properties that are in a tree view, I will use the notation structure *Size:Width,* which refers to the *Size* property and the *Width* attribute.

4. Make sure you have selected the Form control named *Form1* as directed in step 3, and then modify the properties using the values in Table 4-1. The property name to modify is located in the left column, and the value to which to set the property is located in the right column. You may have already completed this step, but to facilitate your data entry, verify that you have sorted the properties in ascending alphabetical order.

Property	Value
Text	My Own Browser
Size:Width	640
Size:Height	480

Table 4-1
Form Properties to Change

TIP

To add a control to a form, you need to perform a drag-and-drop operation. This means you'll move your mouse pointer to the Toolbox, drag the desired control to the designer surface, and drop the control onto it.

You'll now add three Windows Forms controls to your browser application: a text box control in which to enter the destination URL, a button to navigate to the Web page, and a WebBrowser control in which the Web page content will be displayed.

5. Drag a WebBrowser control to the designer surface. The WebBrowser control is located in the Toolbox on the left side of the IDE; it's the last control in the Common Controls section. By default, this control will fill the designer surface entirely. Because you don't want that behavior for this particular application, you'll click the black triangle shown here, which will produce the content of a Smart Tag.

   In this particular example, the Smart Tag helps you undock the control from its parent container (the form). Click the Smart Tag, and select Undock in the Parent Container.

6. Expand the control so that it occupies almost the entire designer space. To do this, click any of the control handles to change its size.

7. Select the WebBrowser control by clicking anywhere on the control. Then go to the Properties window, and modify the values for the properties listed in Table 4-2. Modify the values in the same way you modified the form controls in step 4.

Property	Value
(Name)	myBrowser
Size:Width	607
Size:Height	385
Location:X	12
Location:Y	12

Table 4-2
WebBrowser Control Properties to Change

8. Drag a text box control and a button control from the Toolbox's Common Controls section to the form so that it looks like Figure 4-2. Change the properties of the controls as you did with the WebBrowser control in step 7. Select one control at a time, and modify its properties with the data in Table 4-3.
9. At this point, you have a complete Web browser—congratulations! You can compile and execute your application by pressing F5.

If you followed the previous steps exactly, your application should now be running. Because we didn't code any functionality, entering a URL and clicking the GO button will not do anything.

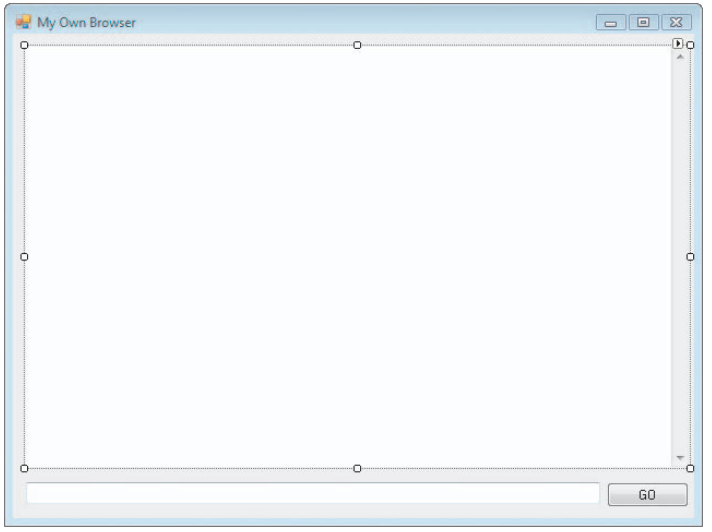


Figure 4-2
MyOwnBrowser application

Control	Property	Value
Textbox	(Name)	tbURL
Textbox	Location:X	12
Textbox	Location:Y	411
Textbox	Size:Width	526
Textbox	Size:Height	20
Button	(Name)	btnGo
Button	Location:X	544
Button	Location:Y	409
Button	Text	GO

Table 4-3
Controls, Properties, and Values

You first have to "wire up" the controls to the functionality that they will perform. I will use an analogy to explain this fundamental concept. A light bulb by itself is not a useful piece of hardware. To obtain light from it, you need to connect two wires carrying electricity. Similar to what an electrician would do to create this electrical circuit, you need to attach, or *wire*, the control and the action together by writing code to handle the event of clicking the GO button. Keep this analogy in mind when you see references to the term *wire* or *wiring* used in this book.

Before we wire up the click action to the button, I'll explain the line of code you'll add in the following instructions, and I'll explain how it relates to the OOP concepts previously introduced in Chapter 1, "Introducing Microsoft Visual C# 2008 Express Edition."

When you dropped the controls onto the designer surface, you created instances of the class represented by those controls. For example, when you dropped the WebBrowser control, you created an instance of the class *System.Windows.Forms.WebBrowser* that you then named *myBrowser*. The *WebBrowser* class has many methods, and the *Navigate* method is the one you'll use. As its name implies, this method allows the *WebBrowser* class to navigate to a URL. The URL is passed as an argument to the *Navigate* method. An argument, also called a *parameter*, is used to pass data to a method.

The argument in this case is the text the user will enter in the instance of the *System.Windows.Forms.TextBox* class that you appropriately named *tbURL*. To retrieve the content of the text box control named *tbURL*, you use the *Text* property of that control. A property enables you to set or retrieve the content of a data member in a class without accessing the data member directly. That way, the provider of the class (for example, Microsoft) can modify the implementation of the *Text* property without concerning the user with the implementation details. In OOP, this is called *encapsulation*. You can compare this process to a person driving a car: you don't need to know how the engine and transmission work to drive the car. Another good example is the *Navigate* method. You don't need to know how it's implemented; you simply want it to do its job. As mentioned earlier, many things are happening when you design a form with Visual Studio. You have seen that you don't need to create any of the classes or instances representing your controls because Visual Studio is doing all of that for you!

TO WIRE THE CLICK ACTION TO A BUTTON

1. Close the running application, and return to the IDE. Double-click the button control. You'll see the code window shown in Figure 4-3.

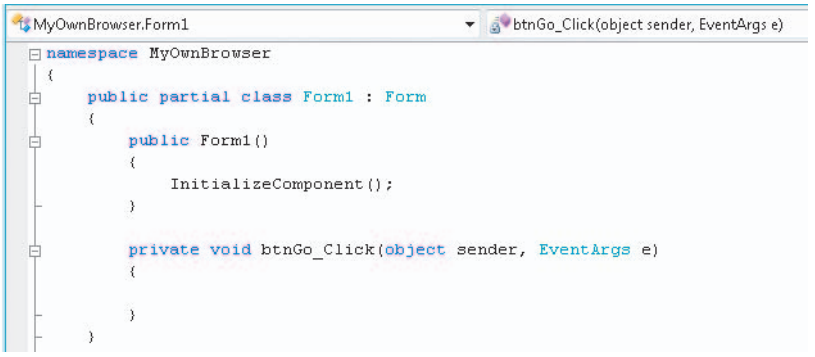


Figure 4-3
Code window for the btnGo_Click *event*

If you terminated the execution of your application properly, you should see the source code window with the *btnGo_Click* event template. When you double-clicked the button control, you signaled to Visual Studio that you wanted to wire the click action to the button control. Typically, each control can trigger multiple events depending on which behavior you want to intercept with your code. Each control has a default event that becomes available to the programmer for coding by double-clicking the control on the designer surface. In this case, Visual Studio created the *Click* event template so that you could enter the following code.

2. Type the following code at the cursor:

```
myBrowser.Navigate(tbURL.Text);
```

3. Press F5 to compile and execute the application. If you named your controls correctly in step 8 in the previous exercise and entered the line of code as shown in step 2 of this exercise, you should now have your own Web browser application that takes you to a Web page when you enter a URL. Of course, you won't have all the bells and whistles of Internet Explorer, but be patient—we're getting there. Try going to your favorite URLs to see whether your browser is working as expected. For instance, I went to *www.microsoft.com*, and it worked just fine! You can see the result in Figure 4-4.

NOTE

If you try to type some code and it doesn't work, your application is probably still running. If you don't close the application and you return to Visual C#, you won't be able to modify the source code. A good way to verify that you have closed and terminated the application is to look in the Visual C# Express Edition title bar. If you see the name of your application followed by the word *(running)*, this means your application is still active and you won't be able to add code. If you try to add code, the status bar will report that you are in read-only mode with the following message: "Cannot currently modify this text in the editor. It is in read-only."

NOTE

Everything in C# is case-sensitive, so *tbURL* and *tbUrl* are two different values.

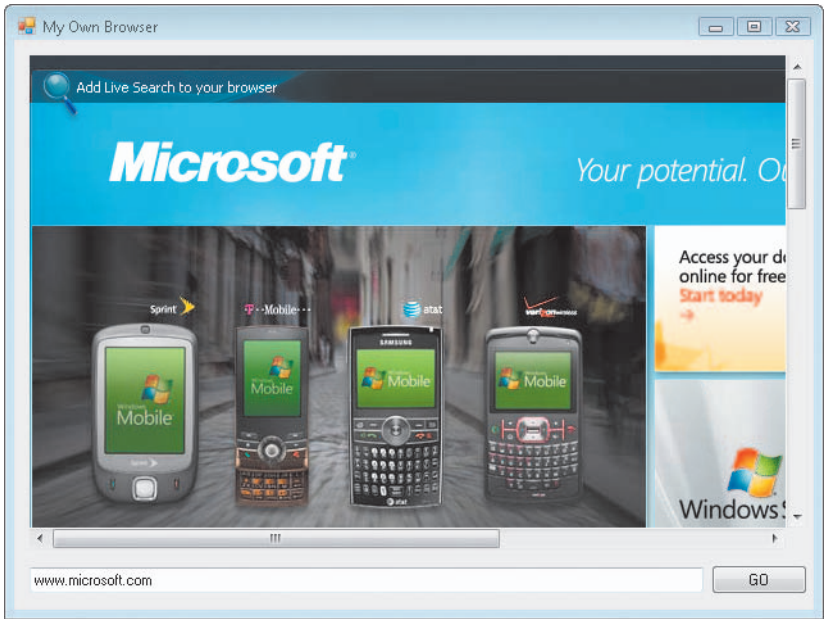


Figure 4-4
MyOwnBrowser showing the Microsoft.com Web site

Putting It All Together

MORE INFO

Philosophies differ when it comes to naming the variable that represents controls on the design surface. In this book, I'll use up to three letters to describe and identify the control type by looking at its name, such as *btn* for a button control. The variable name then becomes *btnGo*. I will introduce the list when I talk about common controls in Chapter 5, "Using Rapid Application Development Tools with Visual C# 2008."

You've just seen that when you drag a control to the design surface, you're actually creating an object of that control class. When you're naming the control in the Properties window, you're actually assigning a name to the variable you've just created—which is exactly what you did for the three controls used in your browser. In fact, this is why you want to give your controls meaningful names so that you can use them later programmatically.

As you now know, a great deal of activity was taking place when you dropped controls on the designer surface. To help you understand what took place in the background, we talked about important OOP concepts behind the line of code you added to respond to the *Click* event.

Now that you've run the application, here is a list of questions you may have:

- What happens if I put nothing in the text box and hit Enter?
- What happens if I enter an invalid URL?
- What happens if I enter anything I want?

My answer to you is simply, "Try it. Try it now." The real deal is that your Web browser will actually behave like any other Web browser and will navigate to whatever URL is typed into the text box. If you don't type anything, clicking the GO button will have no effect. If you type something that isn't a URL, the browser control will come back with a Page Not Found or Code 404 page.

Now is your time to experiment. Remember this book's rule: try, try, try. Play with it. Change some of the properties, and see the results at run time. Although we haven't used many features yet, you'll add more in Chapter 6. This project is far from over! By adding new features, you'll arrive at a point where your application will start to look much more familiar.

IMPORTANT

Before moving on, I invite you to look at a video on the MSDN Web site that talks about object-oriented programming. You've read a good introduction to OOP both in this chapter and in Chapter 1. To understand the concept from another angle, navigate to *http://msdn2.microsoft.com/en-us/beginner/bb308750.aspx* and view Lesson 6.

Links to More Information

Other good sources of information are the videos from MSDN that were created to cover Visual C# 2005 Express Edition but are still applicable. The videos for Lessons 2 and 7 cover some of the topics you have just learned and will provide you with another point of view. You can find the videos for Lessons 2 and 7 by visiting the following hyperlinks:

For Lesson 2: http://msdn2.microsoft.com/en-us/beginner/bb308738.aspx

For Lesson 7: http://msdn2.microsoft.com/en-us/beginner/bb308753.aspx

In Summary...

In this chapter, you learned how to build a Web browser; in the process, you did the following:

- You added more than one control to the designer surface.
- You set properties in the Properties window.
- You wired an event to a control and learned how to add code that will execute when the event is triggered.

In this example, you saw many OOP concepts in action by using only one line of code. You added the code to respond to the button click event by calling the *Navigate* method of your Web browser object. Your Web browser navigated to a URL passed in as an argument to the *Navigate* method. The argument for the *Navigate* method was passed in using the text box control's *Text* property. Everything was completed and fully working just by tweaking some properties and adding only one line of code! That's what I call productivity.

In the next chapter, you'll continue this process by learning more about the major features of Visual C# 2008 Express Edition. You'll become more productive at developing applications by learning about features such as IntelliSense, snap lines, code snippets, Smart Tags, refactoring, and much more.

Index

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X