

James Foxall


STARTER KIT

DVD includes
Visual C#® 2008
Express Edition

Sams **Teach Yourself**

Visual C#® 2008

in **24**
Hours

SAMS



Sams Teach Yourself Visual C#® 2008 in 24 Hours: Complete Starter Kit

Copyright © 2008 by Pearson Education, Inc.

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, v1.0 or later (the latest version is presently available at <http://www.opencontent.org/openpub/>).

ISBN-13: 978-0-672-32990-6

ISBN-10: 0-672-32990-5

Library of Congress Cataloging-in-Publication data is on file

Printed in the United States of America

First Printing June 2008

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Visual C# is a registered trademark of Microsoft Corporation.

Warning and Disclaimer

Every effort has been made to make this book as complete and accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the CD or programs accompanying it.

Bulk Sales

Sams Publishing offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales

1-800-382-3419

corpsales@pearsontechgroup.com

For sales outside the U.S., please contact

International Sales

international@pearson.com

Editor-in-Chief

Karen Gettman

Executive Editor

Neil Rowe

Development Editor

Mark Renfrow

Managing Editor

Patrick Kanouse

Senior Project Editor

Tonya Simpson

Copy Editor

Margo Catts

Indexer

Tim Wright

Proofreader

Kathy Ruiz

Technical Editor

Todd Meister

Publishing Coordinator

Cindy Teeters

Multimedia Developer

Dan Scherf

Book Designer

Gary Adair



The Safari® Enabled icon on the cover of your favorite technology book means the book is available through Safari Bookshelf. When you buy this book, you get free access to the online edition for 45 days.

Safari Bookshelf is an electronic reference library that lets you easily search thousands of technical books, find code samples, download chapters, and access technical information whenever and wherever you need it.

To gain 45-day Safari Enabled access to this book:

- ▶ Go to <http://www.informit.com/onlineedition>.
- ▶ Complete the brief registration form.
- ▶ Enter the coupon code **RMRP-Y6IP-KPI6-NBPM-YPU7**.

If you have difficulty registering on Safari Bookshelf or accessing the online edition, please email customer-service@safaribooksonline.com.

Introduction

With Microsoft's introduction of the .NET platform, a new, exciting programming language was born. Visual C# is now the language of choice for developing on the .NET platform, and Microsoft has even written a majority of the .NET Framework using Visual C#. Visual C# is a modern object-oriented language designed and developed from the ground up with a best-of-breed mentality, implementing and expanding on the best features and functions found in other languages. Visual C# 2008 combines the power and flexibility of C++ with some of the simplicity of Visual C#.

Audience and Organization

This book is targeted toward those who have little or no programming experience or who might be picking up Visual C# as a second language. The book has been structured and written with a purpose: to get you productive as quickly as possible. I've used my experiences in writing applications with Visual C# and teaching Visual C# to create a book that I hope cuts through the fluff and teaches you what you need to know. All too often, authors fall into the trap of focusing on the technology rather than on the practical application of the technology. I've worked hard to keep this book focused on teaching you practical skills that you can apply immediately toward a development project. Feel free to post your suggestions or success stories at www.jamesfoxall.com/forums.

This book is divided into five parts, each of which focuses on a different aspect of developing applications with Visual C# 2008. These parts generally follow the flow of tasks you'll perform as you begin creating your own programs with Visual C# 2008. I recommend that you read them in the order in which they appear.

- ▶ Part I, "The Visual C# 2008 Environment," teaches you about the Visual C# environment, including how to navigate and access Visual C#'s numerous tools. In addition, you'll learn about some key development concepts such as objects, collections, and events.
- ▶ Part II, "Building a User Interface," shows you how to build attractive and functional user interfaces. In this part, you'll learn about forms and controls—the user interface elements such as text boxes and list boxes.
- ▶ Part III, "Making Things Happen: Programming," teaches you the nuts and bolts of Visual C# 2008 programming—and there's a lot to learn. You'll discover how to create classes and procedures, as well as how to store data, perform loops, and make

Sams Teach Yourself Visual C# 2008 in 24 Hours

decisions in code. After you've learned the core programming skills, you'll move into object-oriented programming and debugging applications.

- ▶ Part IV, "Working with Data," introduces you to working with graphics, text files, and programming databases, and shows you how to automate external applications such as Word and Excel. In addition, this part teaches you how to manipulate a user's file system and the Windows Registry.
- ▶ Part V, "Deploying Solutions and Beyond," shows you how to distribute an application that you've created to an end user's computer. In Hour 24, "The 10,000-Foot View," you'll learn about Microsoft's .NET initiative from a higher, less-technical level.

Many readers of previous editions have taken the time to give me input on how to make this book better. Overwhelmingly, I was asked to have examples that build on the examples in the previous chapters. In this book, I have done that as much as possible. Now, instead of learning concepts in isolated bits, you'll be building a feature-rich Picture Viewer program throughout the course of this book. You'll begin by building the basic application. As you progress through the chapters, you'll add menus and toolbars to the program, build an Options dialog box, modify the program to use the Windows Registry and a text file, and even build a setup program to distribute the application to other users. I hope you find this approach beneficial in that it enables you to learn the material in the context of building a real program.

Conventions Used in This Book

This book uses several design elements and conventions to help you prioritize and reference the information it contains:

By the Way

By the Way boxes provide useful sidebar information that you can read immediately or circle back to without losing the flow of the topic at hand.

Did you Know?

Did You Know? boxes highlight information that can make your Visual C# programming more effective.

Watch Out!

Watch Out! boxes focus your attention on problems or side effects that can occur in specific situations.

New terms appear *italic* for emphasis.

In addition, this book uses various typefaces to help you distinguish code from regular English. Code is presented in a monospace font. Placeholders—words or characters that represent the real words or characters you would type in code—appear in *italic monospace*. When you are asked to type or enter text, that text appears in **bold**.

Some code statements presented in this book are too long to appear on a single line. In these cases, a line-continuation character (an underscore) is used to indicate that the following line is a continuation of the current statement.

Onward and Upward!

This is an exciting time to be learning how to program. It's my sincerest wish that when you finish this book, you feel capable of creating, debugging, and deploying modest Visual C# programs, using many of Visual C#'s tools. Although you won't be an expert, you'll be surprised at how much you've learned. And I hope this book will help you determine your future direction as you proceed down the road to Visual C# mastery.

HOUR 1

Jumping In with Both Feet: A Visual C# 2008 Programming Tour

What You'll Learn in This Hour:

- ▶ Building a simple (yet functional) Visual C# application
- ▶ Letting a user browse a hard drive
- ▶ Displaying a picture from a file on disk
- ▶ Getting familiar with some programming lingo
- ▶ Learning about the Visual Studio .NET IDE

Learning a new programming language can be intimidating. If you've never programmed before, the act of typing seemingly cryptic text to produce sleek and powerful applications probably seems like a black art, and you might wonder how you'll ever learn everything you need to know. The answer is, of course, one step at a time. The first step to learning a language is the same as that of any other activity: *building confidence*. Programming is part art and part science. Although it might seem like magic, it's more akin to illusion: After you know how things work a lot of the mysticism goes away, freeing you to focus on the mechanics necessary to produce any given desired result.

Producing large, commercial solutions is accomplished by way of a series of small steps. After you've finished creating the project in this hour, you'll have a feel for the overall development process and will have taken the first step toward becoming an accomplished programmer. In fact, you will be building upon this Picture Viewer program in subsequent chapters. By the time you complete this book, you will have built a distributable application, complete with resizable screens, an intuitive interface including menus and toolbars, and robust code with professional error handling. But I'm getting ahead of myself!

In this hour, you'll complete a quick tour that takes you step by step through creating a complete, albeit small, Visual C# program. Most introductory programming books start out with the reader creating a simple Hello World program. I've yet to see a Hello World program that's the least bit helpful (they usually do nothing more than print `hello world` to the screen—oh, what fun). So, instead, you'll create a picture viewer application that lets you view Windows bitmaps and icons on your computer. You'll learn how to let a user browse for a file and how to display a selected picture file on the screen. The techniques you learn in this chapter will come in handy in many real-world applications that you'll create, but the goal of this chapter is for you to realize just how much fun it is to program with Visual C#.

Starting Visual C# 2008

Before you begin creating programs in Visual C# 2008, you should be familiar with the following terms:

- ▶ **Distributable component**—The final, compiled version of a project. Components can be distributed to other people and other computers, and they don't require the Visual C# 2008 development environment (the tools you use to create a .NET program) to run (although they do require the .NET runtime, which I discuss in Hour 23, "Deploying Applications"). Distributable components are often called *programs*. In Hour 23, you'll learn how to distribute the Picture Viewer program that you're about to build to other computers.
- ▶ **Project**—A collection of files that can be compiled to create a distributable component (program). There are many types of projects, and complex applications might consist of multiple projects, such as a Windows application project, and support dynamic link library (DLL) projects.
- ▶ **Solution**—A collection of projects and files that make up an application or component.

By the Way

Visual C# is part of a larger entity known as the *.NET Framework*. The .NET Framework encompasses all the .NET technology, including Visual Studio .NET (the suite of development tools) and the Common Language Runtime (CLR), which is the set of files that make up the core of all .NET applications. You'll learn about these items in more detail as you progress through this book. For now, realize that Visual C# is one of many languages that exist within the .NET family. Many other languages, such as Visual Basic, are also .NET languages, make use of the CLR, and are developed within Visual Studio .NET.

Visual Studio 2008 is a complete development environment, and it's called the *IDE* (short for *integrated development environment*). The IDE is the design framework in which you build applications; every tool you'll need to create your Visual C# projects is accessed from within the Visual C# IDE. Again, Visual Studio 2008 supports development in many different languages—Visual C# being one of the most popular. The environment itself is not Visual C#, but the language you use within Visual Studio 2008 is Visual C#. To work with Visual C# projects, you first start the Visual Studio 2008 IDE.

Start Visual Studio 2008 now by choosing Microsoft Visual C# 2008 Express Edition on your Start/Programs menu. If you are running the full retail version of .NET, your shortcut may have a different name. In this case, locate the shortcut on your Start menu and click it once to start the Visual Studio .NET IDE.

Creating a New Project

When you first start Visual Studio .NET, you're shown the Start Page tab within the IDE. You can open projects created previously or create new projects from this Start page (see Figure 1.1). For this quick tour, you're going to create a new Windows application, so open the File menu and click New Project to display the New Project dialog box shown in Figure 1.2.

If your Start page doesn't look like the one in Figure 1.1, chances are that you've changed the default settings. In Hour 2, "Navigating Visual C# 2008," I'll show you how to change them back.

**By the
Way**

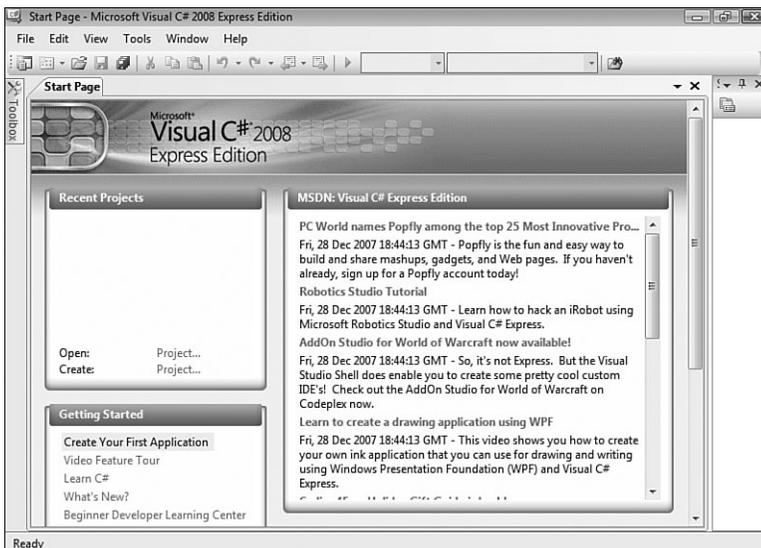
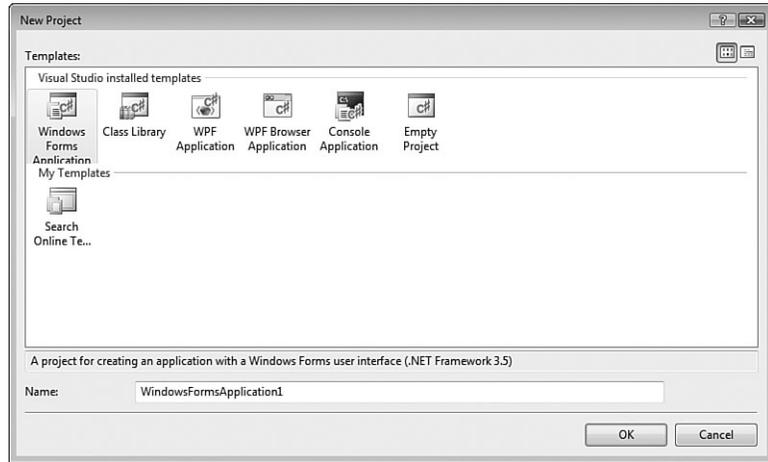


FIGURE 1.1
You can open existing projects or create new projects from the Visual Studio Start page.

FIGURE 1.2

The New Project dialog box enables you to create many types of .NET projects.



The New Project dialog box is used to specify the type of Visual C# project to create. (You can create many types of projects with Visual C#, as well as with the other supported languages of the .NET Framework.) The options shown in Figure 1.2 are limited because I am running the Express edition of Visual C# for all examples in this book. If you are running the full version of Visual C#, many more options are available to you.

Create a new Windows application by following these steps:

1. Make sure that the Windows Application icon is selected (if it's not, click it once to select it).
2. At the bottom of the New Project dialog box is a Name text box. This is where, oddly enough, you specify the name of the project you're creating. Enter *Picture Viewer* in the Name text box.
3. Click OK to create the project.

Did you Know?

Always set the Name text box to something meaningful before creating a project, or you'll have more work to do later if you want to move or rename the project.

When Visual C# creates a new Windows application project, it adds one form (the empty gray window) for you to begin building the *interface*—the graphical windows with which you interact—for your application (see Figure 1.3).

Within Visual Studio 2008, form is the term given to the design-time view of windows that can be displayed to a user.

**By the
Way**

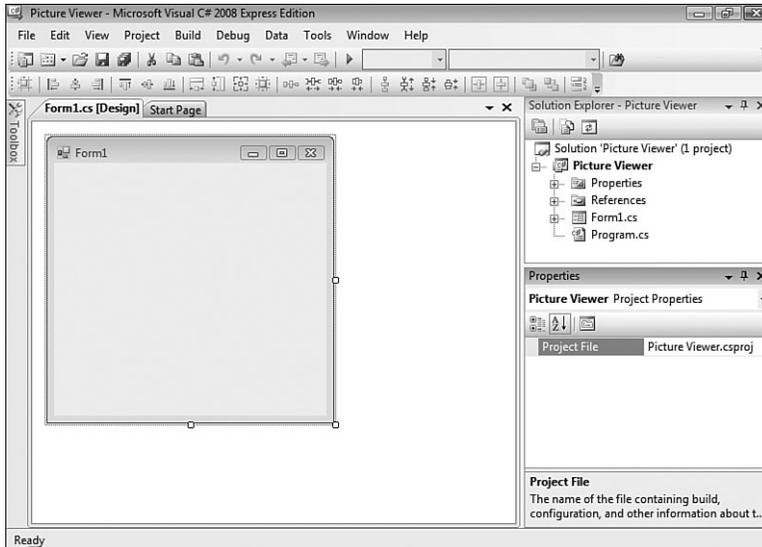


FIGURE 1.3
New Windows applications start with a blank form; the fun is just beginning!

Your Visual Studio 2008 environment might look different from that shown in the figures of this hour because of the edition of Visual Studio 2008 you're using, whether you've already played with Visual Studio 2008, and other factors such as the resolution of your monitor. All the elements discussed in this hour exist in all editions of Visual Studio 2008, however. (If a window shown in a figure isn't displayed in your IDE, use the View menu to display it.)

To create a program that can be run on another computer, you start by creating a project and then compiling the project into a component such as an *executable* (a program a user can run) or a *DLL* (a component that can be used by other programs and components). The compilation process is discussed in detail in Hour 23, "Deploying Applications." The important thing to note at this time is that when you hear someone refer to *creating* or *writing* a program, just as you're creating the Picture Viewer program now, they're referring to the completion of all steps up to and including compiling the project to a distributable file.

**By the
Way**

Understanding the Visual Studio .NET Environment

The first time you run Visual Studio 2008, you'll notice that the IDE contains a number of windows, such as the Solutions Explorer window on the right, which is used to view the files that make up a project. In addition to these windows, the IDE contains a number of tabs, such as the vertical Toolbox tab on the left edge of the IDE (refer to Figure 1.3). Try this now: Click the Toolbox tab to display the Toolbox window (clicking a tab displays an associated window). You can hover the mouse over a tab for a few seconds to display the window as well. To hide the window, simply move the mouse off the window (if you hovered over the tab to display it) or click on another window. To close the window completely, click the Close (X) button in the window's title bar.

By the Way

If you opened the toolbox by clicking its tab rather than hovering over the tab, the toolbox does not automatically close. Instead, it stays open until you click on another window.

You can adjust the size and position of any of these windows, and you can even hide and show them as needed. You'll learn how to customize your design environment in Hour 2.

Watch Out!

Unless specifically instructed to do so, don't double-click anything in the Visual Studio 2008 design environment. Double-clicking most objects produces an entirely different result than single-clicking does. If you mistakenly double-click an object on a form (discussed shortly), a code window is displayed. At the top of the code window is a set of tabs: one for the form design and one for the code. Click the tab for the form design to hide the code window and return to the form.

The Properties window at the right side of the design environment is perhaps the most important window in the IDE, and it's the one you'll use most often. If your computer display resolution is set to 800×600, you can probably see only a few properties at this time. This makes it difficult to view and set properties as you create projects. All the screen shots in this book are taken at 800×600 due to size constraints, but you should run at a higher resolution if you can. I highly recommend that you develop applications with Visual C# at a screen resolution of 1024×768 or higher because it offers plenty of work space. Keep in mind, however, that end users might be running at a lower resolution than you are using for development. If you need to change your display settings, right-click your desktop and select Personalize.

Changing the Characteristics of Objects

Almost everything you work with in Visual C# is an object. Forms, for instance, are objects, as are all the items you can put on a form to build an interface such as list boxes and buttons. There are *many* types of objects, and objects are classified by type. For example, a form is a Form object, whereas items you can place on a form are called Control objects, or controls. (Hour 3, “Understanding Objects and Collections,” discusses objects in detail.) Some objects don’t have a physical appearance but exist only in code, and you’ll learn about these kinds of objects in later hours.

You’ll find that I often mention material coming up in future chapters. In the publishing field, we call these *forward references*. For some reason, these tend to really unnerve some people. I do this only so that you realize you don’t have to fully grasp a subject when it’s first presented; the material is covered in more detail later. I try to keep forward references to a minimum, but teaching programming is, unfortunately, not a perfectly linear process. There will be times I’ll have to touch on a subject that I feel you’re not ready to dive into fully yet. When this happens, I give you a forward reference to let you know that the subject is covered in greater detail later on.

**Watch
Out!**

Every object has a distinct set of attributes known as *properties* (regardless of whether the object has a physical appearance). You have certain properties about you, such as your height and hair color. Visual C# objects have properties as well, such as Height and BackColor. Properties define an object’s characteristics. When you create a new object, the first thing you need to do is set its properties so that the object appears and behaves the way you want it to. To display an object’s properties, click the object in its designer (the main work area in the IDE).

First, make sure your Properties Window is displayed by opening the View menu and choosing Properties Window. Next, click anywhere in the default form now (its title bar says Form1) and check to see whether its properties are displayed in the Properties window. You’ll know because the drop-down list box at the top of the properties window contains the form’s name: Form1 System.Windows.Forms.Form. Form1 is the name of the object, and System.Windows.Forms.Form is the type of object.

Naming Objects

The property you should always set first for any new object is the Name property. Scroll toward the top of the properties list until you see the (Name) property (see Figure 1.4). If the Name property isn’t one of the first properties listed, your properties

window is set to show properties categorically instead of alphabetically. You can show the list alphabetically by clicking the Alphabetical button that appears just above the properties grid.

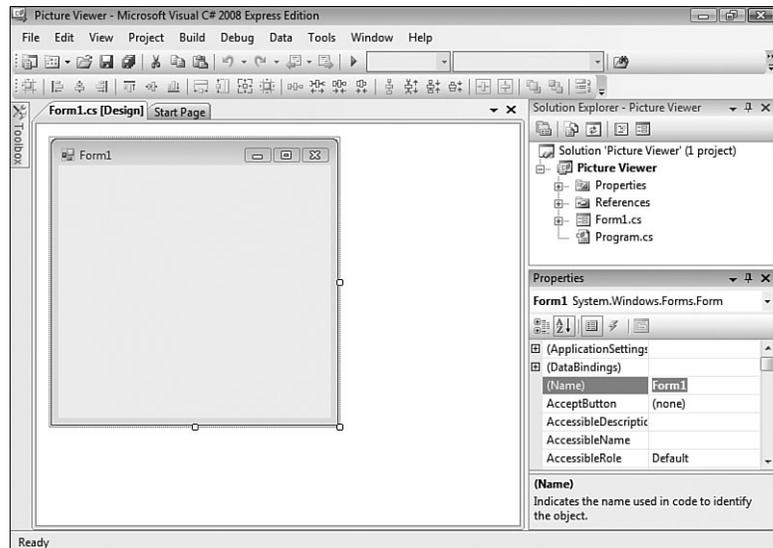
By the Way

I recommend that you keep the Properties window set to show properties in alphabetical order; doing so makes it easier to find properties that I refer to in the text. Note that the Name property always stays toward the top of the list and is referred to as (Name). If you're wondering why it has parentheses around it, that's because the parentheses force the property to the top of the list because symbols come before letters in an alphabetical sort.

When saving a project, you choose a name and a location for the project and its files. When you first create an object, Visual C# gives the object a unique, generic name based on the object's type. Although these names are functional, they simply aren't descriptive enough for practical use. For instance, Visual C# named your form Form1, but it's common to have dozens of forms in a project, and it would be extremely difficult to manage such a project if all forms were distinguishable only by a number (Form2, Form3, and so forth).

FIGURE 1.4

The Name property is the first property you should change when you add a new object to your project.



By the Way

What you're actually working with is a *form class*, or *template*, that will be used to create and show forms at runtime. For the purpose of this quick tour, I simply refer to it as a form. See Hour 5, "Building Forms—The Basics," for more information.

To better manage your forms, give each one a descriptive name. Visual C# gives you the chance to name new forms as they're created in a project. Visual C# created this default form for you, so you didn't get a chance to name it. It's important to not only change the form's name but also to change its filename. Change the program-name and the filename at the same time by following these steps:

1. Click the Name property and change the text from Form1 to ViewerForm. Notice that this does not change the form's filename as it's displayed in the Solution Explorer window located above the Properties window.
2. Right-click Form1.cs in the Solution Explorer window (the window above the properties window).
3. Choose Rename from the context menu that appears.
4. Change the text from Form1.cs to **ViewerForm.cs**.

I use the Form suffix here to denote that the file is a form class. Suffixes are optional, but I find they really help you keep things organized.

**By the
Way**

The Name property of the form is actually changed for you automatically when you rename the file. I had you explicitly change the Name property because it's something you're going to be doing a lot—for all sorts of objects.

Setting the Text Property of the Form

Notice that the text that appears in the form's title bar says Form1. Visual C# sets the form's title bar to the name of the form when it's first created but doesn't change it when you change the form's name. The text in the title bar is determined by the value of the Text property of the form. Change the text now by following these steps:

1. Click the form once more so that its properties appear in the Properties window.
2. Use the scrollbar in the Properties window to locate the Text property.
3. Change the text to **Picture Viewer**. Press the Enter key or click on a different property. You'll see the text in the title bar of the form change.

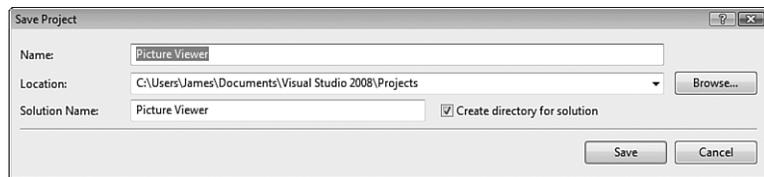
Saving a Project

The changes you've made so far exist only in memory; if you were to turn off your computer at this time, you would lose all your work up to this point. Get into the habit of frequently saving your work, which commits your changes to disk.

Click the Save All button on the toolbar (the picture of a stack of disks) now to save your work. Visual C# then displays the Save Project dialog box shown in Figure 1.5. Notice that the Name property is already filled in because you named the project when you created it. The Location text box is where you specify the location in which to save the project. Visual C# creates a subfolder in this location, using the value in the Name text box (in this case, Picture Viewer). You can use the default location, or change it to suit your purposes. You can have Visual C# create a solution folder in which the project folder gets placed. On large projects, this is a handy feature. For now, it's an unnecessary step, so uncheck the Create Directory for Solution box and then click Save to save the project.

FIGURE 1.5

When saving a project, choose a name and a location for the project and its files.



Giving the Form an Icon

Everyone who has used Windows is familiar with icons—the little pictures that represent programs. Icons most commonly appear in the Start menu next to the names of their respective programs. In Visual C#, you not only have control over the icon of your program file, you can also give every form in your program a unique icon if you want to.

By the Way

The following instructions assume that you have access to the source files for the examples in this book. They are available at www.sampublishing.com. You can also get these files, as well as discuss this book, at my website at <http://www.jamesfoxall.com/books.aspx>. When you unzip the samples, a folder is created for each hour, and within each hour's folder are subfolders for the sample projects. You can find the icon in the folder Hour 1\Picture Viewer.

You don't have to use the icon I've provided for this example; you can use any icon of your choice. If you don't have an icon available (or you want to be a rebel), you can skip this section without affecting the outcome of the example.

To give the form an icon, follow these steps:

1. In the Properties window, click the Icon property to select it.
2. When you click the Icon property, a small button with three dots appears to the right of the property. Click this button.

3. Use the Open dialog box that appears to locate the `PictureViewer.ico` file or another icon file of your choice. When you've found the icon, double-click it, or click it once to select it and then click Open.

After you've selected the icon, it appears in the `Icon` property along with the word "Icon." A small version of the icon appears in the upper-left corner of the form as well. Whenever this form is minimized, this is the icon displayed on the Windows taskbar.

Changing the Size of the Form

Next, you're going to change the `Width` and `Height` properties of the form. The `Width` and `Height` values are shown collectively under the `Size` property; `Width` appears to the left of the comma, `Height` to the right. You can change the `Width` or `Height` property by changing the corresponding number in the `Size` property. Both values are represented in pixels (that is, a form that has a `Size` property of `200,350` is 200 pixels wide and 350 pixels tall). To display and adjust the `Width` and `Height` properties separately, click the small plus sign (+) next to the `Size` property (see Figure 1.6).

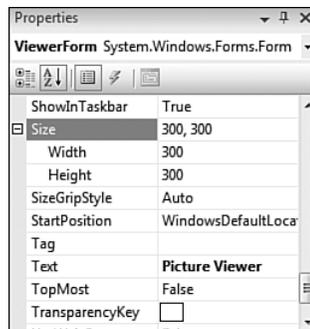


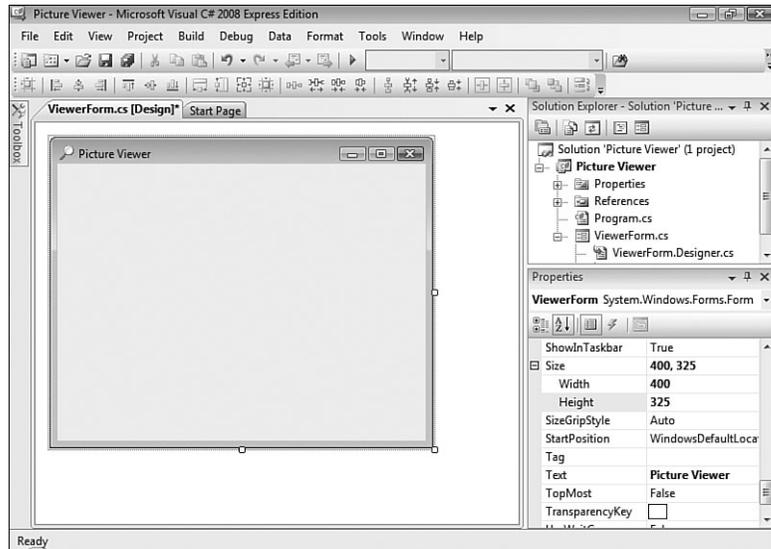
FIGURE 1.6 Some properties can be expanded to show more specific properties.

A pixel is a unit of measurement for computer displays; it's the smallest visible "dot" on the screen. The resolution of a display is always given in pixels, such as `800×600` or `1024×768`. When you increase or decrease a property by one pixel, you're making the smallest possible visible change to the property.

**By the
Way**

Change the Width property to 400 and the Height to 325 by typing in the corresponding box next to a property name. To commit a property change, press Tab or Enter, or click a different property or window. Your screen should now look like the one in Figure 1.7.

FIGURE 1.7
Changes made in the Properties window are reflected as soon as they're committed.



By the Way

You can also size a form by dragging its border, which you'll learn about in Hour 2. This property can also be changed by program code, which you'll learn how to write in Hour 5.

Save the project now by choosing File, Save All from the menu or by clicking the Save All button on the toolbar—it has a picture of stacked disks on it.

Adding Controls to a Form

Now that you've set your form's initial properties, it's time to create a user interface by adding objects to the form. Objects that can be placed on a form are called *controls*. Some controls have a visible interface with which a user can interact, whereas others are always invisible to the user. You'll use controls of both types in this example. On the left side of the screen is a vertical tab titled Toolbox. Click the Toolbox tab now to display the Toolbox window and click the plus sign next to Common Controls to see the most commonly used controls (see Figure 1.8). The toolbox contains all the controls available in the project, such as labels and text boxes.

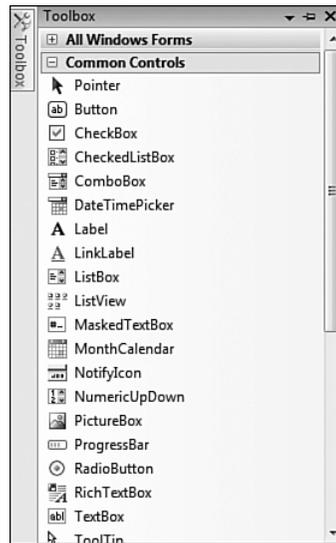


FIGURE 1.8
The toolbox is used to select controls to build a user interface.

The toolbox closes as soon as you've added a control to a form and when the pointer is no longer over the toolbox. To make the toolbox stay visible, click the little picture of a pushpin located in the toolbox's title bar.

I don't want you to add them yet, but your Picture Viewer interface will consist of the following controls:

- ▶ **Two Button controls**—The standard buttons that you're used to clicking in pretty much every Windows program you've ever run
- ▶ **A PictureBox control**—A control used to display images to a user
- ▶ **An OpenFileDialog control**—A hidden control that exposes the Windows Open File dialog box functionality

Designing an Interface

It's generally best to design a form's user interface and then add the code behind the interface to make the form functional. You'll build your interface in the following sections.

Adding a Visible Control to a Form

Start by adding a Button control to the form. Do this by double-clicking the Button item in the toolbox. Visual C# creates a new button and places it in the upper-left corner of the form (see Figure 1.9).

FIGURE 1.9

When you double-click a control in the toolbox, the control is added to the upper-left corner of the form.



Using the Properties window, set the button's properties as follows. Remember, when you view the properties alphabetically, the Name property is listed first, so don't go looking for it down in the list or you'll be looking awhile.

Property	Value
Name	btnSelectPicture
Location	295, 10 (Note: 295 is the x coordinate; 10 is the y coordinate.)
Size	85, 23
Text	Select Picture

You're now going to create a button that the user can click to close the Picture Viewer program. Although you could add another new button to the form by double-clicking the Button control on the toolbox again, this time you'll add a button to the form by creating a copy of the button you've already defined. This enables you to easily create a button that maintains the size and other style attributes of the original button when the copy was made.

To do this, right-click the Select Picture button and choose Copy from its shortcut menu. Next, right-click anywhere on the form and choose Paste from the form's shortcut menu (you could have also used the keyboard shortcuts Ctrl+C to copy and Ctrl+V to paste). The new button appears centered on the form, and it's selected by

default. Notice that it retained almost all of the properties of the original button, but the name has been reset. Change the new button's properties as follows:

Property	Value
Name	btnQuit
Location	295, 40
Text	Quit

The last visible control you need to add to the form is a PictureBox control. A PictureBox has many capabilities, but its primary purpose is to show pictures, which is precisely what you'll use it for in this example. Add a new PictureBox control to the form by double-clicking the PictureBox item in the toolbox and set its properties as follows:

Property	Value
Name	picShowPicture
BorderStyle	FixedSingle
Location	8, 8
Size	282, 275

After you've made these property changes, your form will look like the one in Figure 1.10. Click the Save All button on the toolbar to save your work.

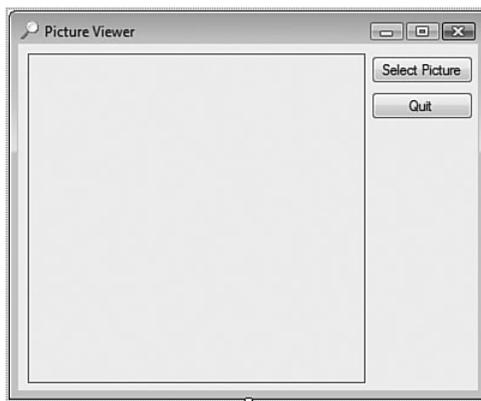


FIGURE 1.10
An application's interface doesn't have to be complex to be useful.

Adding an Invisible Control to a Form

All the controls that you've used so far sit on a form and have a physical appearance when the application is run by a user. Not all controls have a physical appearance,

however. Such controls, referred to as *nonvisual controls* (or *invisible-at-runtime controls*), aren't designed for direct user interactivity. Instead, they're designed to give you, the programmer, functionality beyond the standard features of Visual C#.

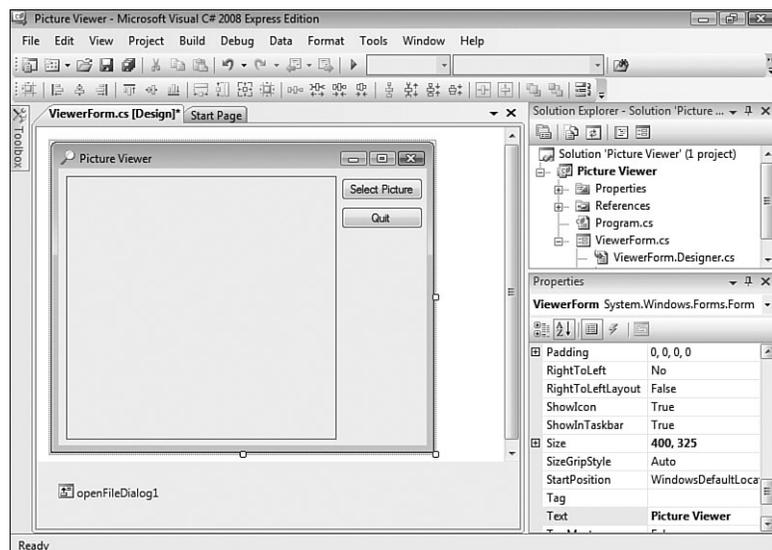
To enable the user to select a picture to display, you need to make it possible to locate a file on a hard drive. You might have noticed that whenever you choose to open a file from within any Windows application, the dialog box displayed is almost always the same. It doesn't make sense to force every developer to write the code necessary to perform standard file operations, so Microsoft has exposed the functionality via a control that you can use in your projects. This control is called the `OpenFileDialog` control, and it will save you dozens and dozens of hours that would otherwise be necessary to duplicate this common functionality.

By the Way

Other controls in addition to the `OpenFileDialog` control give you file functionality. For example, the `SaveFileDialog` control provides features for enabling the user to specify a filename and path for saving a file.

Display the toolbox now and scroll down (using the down arrow in the lower part of the toolbox) until you can see the `OpenFileDialog` control (it's in the Dialogs category), and then double-click it to add it to your form. Note that the control isn't placed on the form, but rather it appears in a special area below the form (see Figure 1.11). This happens because the `OpenFileDialog` control has no form interface to display to a user. It does have an interface (a dialog box) that you can display as necessary, but it has nothing to display directly on a form.

FIGURE 1.11
Controls that have no interface appear below the form designer.



Select the OpenFileDialog control and change its properties as follows:

Property	Value
Name	ofdSelectPicture
Filename	<make empty>
Filter	Windows Bitmaps *.BMP JPEG Files *.JPG
Title	Select Picture

Don't actually enter the text **<make empty>** for the filename; I really mean delete the default value and make this property value empty.

**By the
Way**

The Filter property is used to limit the types of files that will be displayed in the Open File dialog box. The format for a filter is *description|filter*. The text that appears before the first pipe symbol is the descriptive text of the file type, whereas the text after the pipe symbol is the pattern to use to filter files. You can specify more than one filter type by separating each description|filter value with another pipe symbol. Text entered into the Title property appears in the title bar of the Open File dialog box.

The graphical interface for your Picture Viewer program is now finished. If you pinned the toolbox open, click the pushpin in the title bar of the toolbox now to close it.

Writing the Code Behind an Interface

You have to write code for the program to be capable of performing tasks and responding to user interaction. Visual C# is an *event-driven* language, which means that code is executed in response to events. These events might come from users, such as a user clicking a button and triggering its Click event, or from Windows itself (see Hour 4, "Understanding Events," for a complete explanation of events). Currently, your application looks nice but it won't do a darn thing. Users can click the Select Picture button until they can file for disability with carpal tunnel syndrome, but nothing will happen because you haven't told the program what to do when the user clicks the button. You can see this for yourself now by pressing F5 to run the project. Feel free to click the buttons, but they don't do anything. When you're finished, close the window you created to return to Design mode.

You're going to write code to accomplish two tasks. First, you're going to write code that lets users browse their hard drives to locate and select a picture file and then display the file in the picture box (this sounds a lot harder than it is). Second, you're

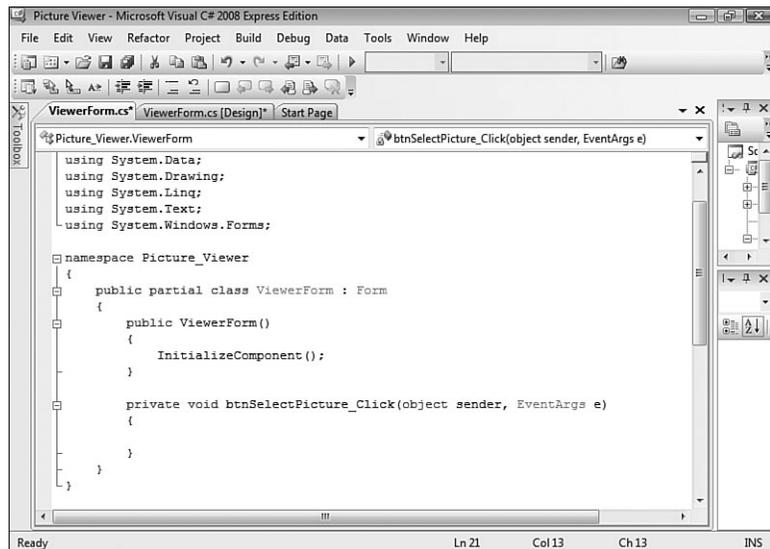
going to add code to the Quit button that shuts down the program when the user clicks the button.

Letting a User Browse for a File

The first bit of code you're going to write enables users to browse their hard drives, select a picture file, and then see the selected picture in the PictureBox control. This code executes when the user clicks the Select Picture button; therefore, it's added to the Click event of that button.

When you double-click a control on a form in Design view, the default event for that control is displayed in a code window. The default event for a Button control is its Click event, which makes sense because clicking is the most common action a user performs with a button. Double-click the Select Picture button now to access its Click event in the code window (see Figure 1.12).

FIGURE 1.12
You'll write all code in a window such as this.



When you access an event, Visual C# builds an event handler, which is essentially a template procedure in which you add the code that executes when the event occurs. The cursor is already placed within the code procedure, so all you have to do is add code. Although this may seem daunting to you now, by the time you're finished with this book you'll be madly clicking and clacking away as you write your own code to make your applications do exactly what you want them to do—well, most of the time. For now, just enter the code as I present it here.

It's important that you get in the habit of commenting your code, so the first statement you're going to enter is a comment. Beginning a statement with two forward slashes designates the statement as a comment; the compiler doesn't do anything with the statement, so you can enter whatever text you want after the two forward slashes. Type the following statement exactly as it appears and press the Enter key at the end of the line:

```
// Show the open file dialog box.
```

The next statement you enter triggers a method of the `OpenFileDialog` control that you added to the form. You'll learn all about methods in Hour 3. For now, think of a method as a mechanism to make a control do something. The `ShowDialog()` method tells the control to show its Open dialog box and let the user select a file. The `ShowDialog()` method returns a value that indicates its success or failure, which you'll then compare to a predefined result (`DialogResult.OK`). Don't worry too much about what's happening here; you'll be learning the details of all this in later hours, and the sole purpose of this hour is to get your feet wet. In a nutshell, the `ShowDialog()` method is invoked to let a user browse for a file. If the user selects a file, more code is executed. Of course, there's a lot more to using the `OpenFileDialog` control than I present in this basic example, but this simple statement gets the job done. Enter the following two code statements, pressing Enter at the end of each line:

Capitalization is important. Visual C# is a *case-sensitive language*, which means `ShowDialog()` is not the same as `Showdialog()`. If you get the case of even one letter wrong, Visual C# doesn't recognize the word and your code doesn't work, so always enter code exactly as it appears in this book!

**By the
Way**

```
if (ofdSelectPicture.ShowDialog() == DialogResult.OK)
{
```

The opening brace (the `{` character) is necessary for this `if` statement because it denotes that this `if` construct will be made up of multiple lines.

Time for another comment. Your cursor is currently on the line below the `{` that you entered. Type this statement and remember to press Enter at the end of the code line.

```
// Load the picture into the picture box.
```

This next statement is the line of code that actually displays the picture in the picture box.

Enter the following statement:

```
picShowPicture.Image = Image.FromFile(ofdSelectPicture.FileName);
```

In addition to displaying the selected picture, your program is also going to display the path and filename of the picture in the title bar. When you first created the form, you changed the form's Text property in the Properties window. To create dynamic applications, properties need to be constantly adjusted at runtime, and this is done with code. Insert the following two statements (press Enter at the end of each line):

```
// Show the name of the file in the form's caption.
this.Text = string.Concat("Picture Viewer(" + ofdSelectPicture.FileName + ")");
```

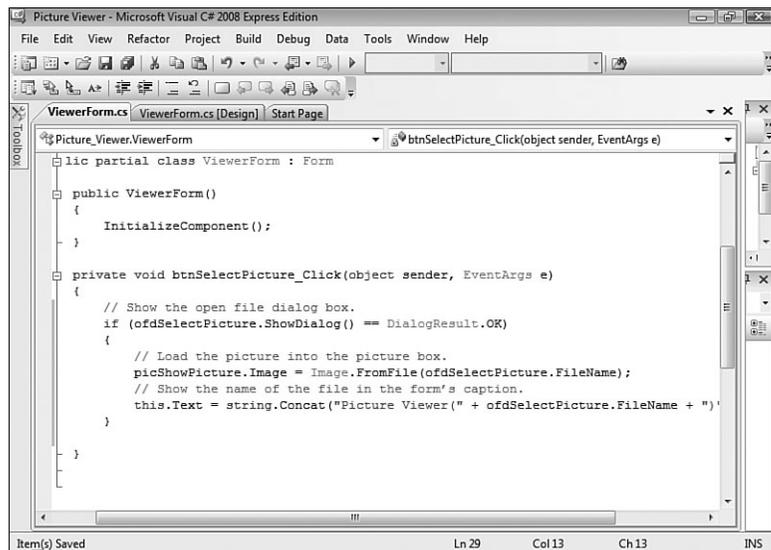
The last statement you need to enter is a closing brace (a } character). Whenever you have an opening brace, you have to have a closing brace. This is how Visual C# groups multiple statements of code. Enter this statement now:

```
}
```

After you've entered all the code, your editor should look like that shown in Figure 1.13.

FIGURE 1.13

Make sure that your code exactly matches the visible code shown here.



Terminating a Program Using Code

The last bit of code you'll write terminates the application when the user clicks the Quit button. To do this, you'll need to access the Click event handler of the btnQuit button. At the top of the code window are two tabs. The current tab has the text ViewerForm.cs*. This is the tab containing the code window for the form with the filename ViewerForm.cs. Next to this is a tab that contains the text

ViewerForm.cs [Design]*. Click this tab now to switch from Code view to the form designer. If you receive an error when you click the tab, the code you entered contains an error, and you need to edit it to make it the same as shown in Figure 1.13. After the form designer appears, double-click the Quit button to access its Click event.

Enter the following code in the Quit button's Click event handler and press Enter at the end of each statement:

```
// Close the window and exit the application
this.Close();
```

The `this.Close();` statement closes the current form. When the last loaded form in a program is closed, the application shuts itself down—completely. As you build more robust applications, you'll probably want to execute all kinds of clean-up routines before terminating an application, but for this example, closing the form is all you need to do.

**By the
Way**

Running a Project

Your application is now complete. Click the Save All button on the toolbar (it looks like a stack of disks), and then run your program by pressing F5. You can also run the program by clicking the button on the toolbar that looks like a right-facing triangle and resembles the Play button on a DVD (this button is called Start, and it can also be found on the Debug menu). Learning the keyboard shortcuts will make your development process move along faster, so I recommend you use them whenever possible.

When you run the program, the Visual C# interface changes, and the form you've designed appears floating over the design environment (see Figure 1.14).

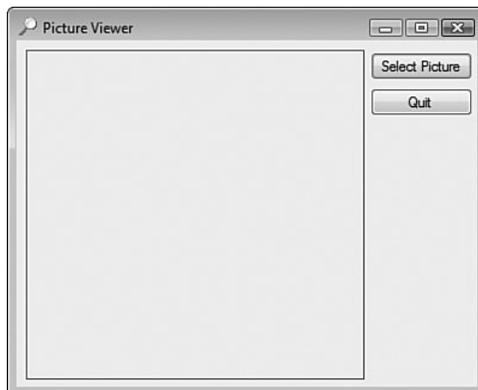


FIGURE 1.14
When in Run mode, your program executes the same as it would for an end user.

You are now running your program as though it were a standalone application running on another user's machine; what you see is exactly what users would see if they ran the program (without the Visual Studio 2008 design environment in the background, of course). Click the Select Picture button to display the Select Picture dialog box (see Figure 1.15). Use the dialog box to locate a picture file. When you've found a file, double-click it, or click once to select it and then click Open. The selected picture is then displayed in the picture box, as shown in Figure 1.16.

By the Way

When you click the Select Picture button, the default path shown depends on the last active path in Windows, so it might be different for you than what is shown in Figure 1.15.

FIGURE 1.15

The `OpenFileDialog` control handles all the details of browsing for files. Cool, huh?

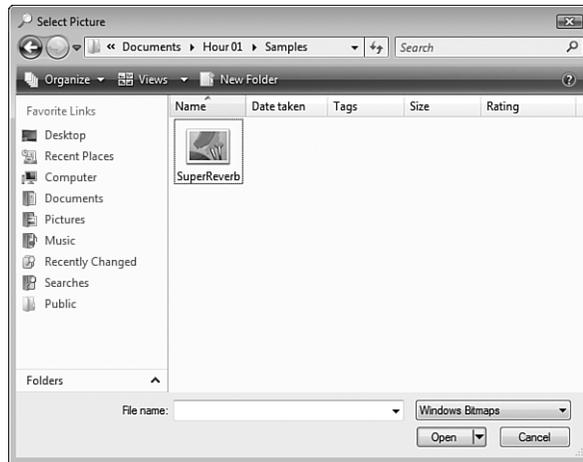


FIGURE 1.16

What could be prettier than a 1964 Fender Super Reverb amplifier?



If you want to select and display a picture from your digital camera, chances are the format is JPEG, so you need to select this from the Files of Type drop-down. Also, if your image is very large, you'll see only the upper-left corner of the image (what fits in the picture box). In later hours, I'll show you how you can scale the image to fit the picture box, and even resize the form to show a larger picture in its entirety.

Summary

When you're finished playing with the program, click the Quit button to return to Design view.

That's it! You've just created a bona fide Visual C# program. You've used the toolbox to build an interface with which users can interact with your program, and you've written code in strategic event handlers to empower your program to do things.

These are the basics of application development in Visual C#. This fundamental approach is used to build even the most complicated programs; you build the interface and add code to make the application do things. Of course, writing code to do things *exactly* the way you want things done is where the process can get complicated, but you're on your way.

If you take a close look at the organization of the hours in this book, you'll see that I start out by teaching you the Visual C# (Visual Studio 2008) environment. I then move on to building an interface, and later I teach you all about writing code. This organization is deliberate. You might be a little anxious to jump in and start writing serious code, but writing code is only part of the equation—don't forget the word *Visual* in Visual C#. As you progress through the hours, you'll be building a solid foundation of development skills.

Soon, you'll pay no attention to the man behind the curtain—you'll *be* that man (or woman)!

Q&A

Q. *Can I show pictures of file types other than BMP and JPG?*

A. Yes. The `PictureBox` control supports the display of images with the extensions BMP, JPG, ICO, EMF, WMF, and GIF. The `PictureBox` control can even save images to a file using any of the supported file types.

Q. *Is it possible to show pictures in other controls?*

- A.** PictureBox is the control to use when you are just displaying images. However, many other controls enable you to display pictures as part of the control. For instance, you can display an image on a button control by setting the button's Image property to a valid picture.

Workshop

The Workshop is designed to help you anticipate possible questions, review what you've learned, and get you thinking about how to put your knowledge into practice.

Quiz

1. What type of Visual C# project creates a standard Windows program?
2. What window is used to change the attributes (location, size, and so on) of a form or control in the IDE?
3. How do you access a control's default event (code)?
4. What property of a picture box do you set to display an image?
5. What is the default event for a button control?

Answers

1. Windows Forms Application
2. The Properties window
3. Double-click the control in the designer
4. The Image property
5. The Click event

Exercises

1. Change your Picture Viewer program so that the user can also locate and select GIF files. (Hint: Change the Filter property of the OpenFileDialog control.)
2. Create a new project with a new form. Create two buttons on the form, one above the other. Next, change their position so that they appear next to each other.

Index

SYMBOLS

- + (addition) operator, 268
- & (ampersands)
 - accelerator keys, 200
 - And operator, 274
- * (asterisks)
 - multiplication operator, 269
 - saving projects, 65
- { } (braces), block statements, 286
- ^ (Xor) operator, 275
- / (division) operator, 269
- = (equal sign), setting properties, 61
- ! (Not) operator, 274
- () (parentheses), methods, 68, 225
- . (periods), writing code, 64
- | (Or) operator, 274
- ; (semicolons), statements, 65
- \ (slashes) as escape sequences, 250
- (subtraction) operator, 269

A

- accelerator keys, 200
- Accept buttons, 160
- AcceptButton property, 160
- ActiveCell object
 - FormulaR1C1 property, 457
- ActiveCell objects, 457
- ActiveMdiChild property, 146
- Add() method
 - Application objects, 462
 - DataTable objects, 448, 450
 - Items collection, 168, 170
 - list boxes, 172
 - List View, 189
 - Tree View control, 192-193
- AddDays method
 - DateTime class, 280
- AddHours method
 - DateTime class, 280

adding

adding

- files
 - to projects, 52-53
- items to lists
 - via code, 189
 - via List View, 187-189
- nodes
 - to tree view, 192-193
- addition (+) operator, 268**
- AddMilliseconds method**
 - DateTime class, 280
- AddMinutes method**
 - DateTime class, 280
- AddMonths method**
 - DateTime class, 280
- AddSeconds method**
 - DateTime class, 280
- AddTwoNumbers() method, 230**
- AddYears method**
 - DateTime class, 280
- ADO.NET, 438**
 - databases
 - closing data source connections, 440
 - connecting to, 438, 440
 - creating records, 448, 450
 - DataAdapter objects, 441-442
 - DataRow objects, 444-445
 - DataTable objects, 441, 446-448, 450
 - deleting records, 450
 - editing records, 448
 - navigating records, 446-448

- running, 451
 - updating records, 448
- Advanced Appearance dialog**
 - system colors
 - changing, 376-377
- aligning**
 - controls, 132
- ampersand (&)**
 - accelerator keys, 200
- Anchor property, 136-137**
- anchoring controls, 135, 137-138**
- And (&) operator, 274**
- Application objects, 455-456, 462**
 - ActiveCell objects, 457
 - Add() method, 462
- Archive flag (file attributes), 406**
- arguments**
 - defining, 234
 - passing, 234
- arithmetic operators, 268**
 - addition (+) operator, 268
 - division (/) operator, 269
 - expressions
 - operator precedence, 270-271
 - modulus arithmetic, 269
 - multiplication (*) operator, 269
 - operator precedence, 270-271
 - subtraction (-) operator, 269
- arrays**
 - declaring, 252
 - defining, 241, 251
 - dimensions of, 254
 - jagged arrays, 255

- multidimensional arrays, 253-254
 - two-dimensional arrays, 253
- variables
 - referencing, 252
- asterisks (*)**
 - saving projects, 65
- AutoCompleteMode property**
 - combo boxes, 174
- AutoCompleteSource property**
 - combo boxes, 174
- automatically hiding design windows, 35, 38**
- automation, 453**
 - clients
 - defining, 453
 - Excel, 459
 - adding cell data, 457-458
 - bold cells, 458
 - creating library references, 454
 - selecting cells, 458
 - server creation, 455-456
 - testing, 459
 - viewing, 456
 - workbook creation, 457
 - servers
 - adding Excel cell data, 457-458
 - bold Excel cells, 458
 - creating Excel workbooks, 457
 - creating instances of, 455-456, 461, 463
 - defining, 453
 - Excel, 459

- Excel server creation, 455-456
- selecting Excel cells, 458
- viewing Excel, 456
- Word server creation, 461, 463
- type libraries
 - creating references to, 454, 460
- Word
 - creating library references, 460
 - server creation, 461, 463
- AutoScroll property**
 - scrollable forms, 142
- AutoScrollMargin property**
 - scrollable forms, 142
- AutoScrollMinSize property**
 - scrollable forms, 142
- AutoSize property**
 - Timer control, 179
- autosizing controls, 135, 137-138**

B

- BackColor property, 44, 105, 377**
- BackgroundImage property, 106-109**
- backgrounds (forms)**
 - adding images to, 106-108
 - changing color, 105
 - removing images from, 108
- Backspace key, erasing code, 65**
- BaseDirectory() method, 431**

binding

- early binding, 344-345
- late binding, 344-345
- objects
 - creating via variable dimensioning, 346
 - variable references, 344-345
- bitmaps, creating Graphic objects, 373-374**
- block scope, 255-256**
- block statements, braces ({ }), 286**
- bold cells (Excel), 458**
- bool data type, 244**
- Boolean logic, 272-273**
 - And (&) operator, 274
 - if statements, 285
 - Not (!) operator, 274
 - Or (|) operator, 274
 - Xor (^) operator, 275
- borders (forms), customizing, 110-112**
- BorderStyle property, 43**
- braces ({ }), block statements, 286**
- break points**
 - actions in, 316-317
 - debugging code, 315
- break statements, breaking loops, 302-303**
- BringToFront method, layering controls, 141**
- browsing**
 - files, 24-25
 - scope, 76
- btnAutomateExcel Click events, 456**

build errors, 312-314

Button control, 83, 377

buttons

- Accept button, 160
- Cancel button, 161
- Click events, 160
- creating, 159
- forms, adding to, 109-110
- message boxes
 - determining which button is clicked, 355-356
 - displaying in, 353
- OK button, 159
- PerformClick method, 160
- Picture Viewer project
 - adding to, 63
 - Draw Border button, 68-72
 - Enlarge button, 63, 66
 - Show Control Names button, 74-75
 - Shrink button, 63, 66
- radio buttons, 164-165
- separators, 212
- toolbars
 - adding to, 210, 212
 - drop-down menus, 214
- Buttons property, MessageBox.Show() function, 352**

C

- calling**
 - methods, 229-231
 - procedures, 229-231
- Cancel buttons, 161**

CancelButton property**CancelButton property, 161**

Caption property,

MessageBox.Show() function, 352case statements, **293-294**case-sensitivity (code statements), **25**

casting

 data types, **245** explicit casting, **245** implicit casting, **245**catch statements, **323-324, 327** Exception objects, Message property, **325** Exception variables, **326**

cells (Excel)

 adding data to, **457-458** bold cells, **458** selecting, **458**character limits (text), setting in text boxes, **157**check boxes, **161-162**checked menu items, creating, **202**Checked property, radio buttons, **165**

CheckFileExists property,

OpenFileDialog control, 399CheckState property, **162**circles, drawing, **381**class modules, project management, **51**

classes

 clients, **336** data/code encapsulation, **334-335** defining, **221, 334** instance members, defining, **221** instantiating objects, **343** binding object references to variables, **344-345** object creation via variable dimensioning, **346** object lifetimes, **347-348** releasing object references, **346-347**

methods

 declaring procedures that do not return values, **224-227** declaring procedures that return values, **227-228**

object interfaces

 client interaction with, **338** custom events in, **338** elements of, **337** exposing functions as methods, **343** methods in, **338** properties in, **338-342** servers, **336** static members, defining, **221****ClassesRoot property, Registry object, 416****Clear() method, 70** Graphics object, **381** Items collection, **170** List View, **190** Tree View control, **194****clearing** items from lists via code, **190** nodes from tree view, **194****click events, 23, 159** buttons, **160** Cancel buttons, **161** Items collection, **168-170** mouse, **364****ClickOnce technology, 469-470** advanced settings, **475** application creation, **471-472** Picture Viewer project installation, **474****clients, 336** defining, **453**

object interfaces

 exposing functions as methods, **343** interaction with, **338** properties, **339-342****Close() method, 119, 440****closed design windows, 35****CLR (Common Language Runtime), 480-481****COBOL, IL code, 481****code**

debugging

 adding comments to code, **310-312** break points, **315** build errors, **312-314** catch statements, **323-325** error handlers, **323-325** finally statements, **323-325** Immediate window, **317-320** Output window, **321** runtime errors, **312-314**

- structured exception handling, 322, 325-329
 - try blocks, 323
 - try statements, 323-325
- encapsulating via classes, 334-335
- erasing, 65
- file properties, retrieving, 407-409
- IL code, 481-482
- IntelliSense, 64
- managed code, defining, 480
- periods (.), 64
- procedures, writing via, 54-55
- simple object build example, 69-72
- unmanaged code, defining, 480
- code statements, writing, 25-26**
- collections (objects), 73-76**
- color**
 - BackColor property, 44
 - form backgrounds, changing in, 105
 - object properties, 45-46
 - system colors
 - assigning, 378
 - changing, 376-377
 - syncing interface colors with user system colors, 377-378
- color drop-down list (Properties window), 46**
- columns**
 - DataRow objects, 444
 - lists, creating in, 187
- Columns property, List View control, 187**
- combo boxes, 166**
 - AutoCompleteMode property, 174
 - AutoCompleteSource property, 174
 - drop-down lists, creating, 172-174
 - DropDownList property, 173
 - DropDownStyle property, 173
 - Insert() method, 172
 - Items collection, 172
 - Items property, 173
 - Sorted property, 172
 - Text property, 173
- CommandBuilder objects, 442**
- comments, adding to code, 310-312**
- comparison operators, 271-272**
- compilers**
 - defining, 242
 - JITers, 482
 - reserved words, determining, 250
- components (distributable), defining, 8**
- concatenation strings, 275**
- ConnectionString property, 439**
- constants**
 - benefits of, 246
 - defining, 241, 246-247
 - Prompt on Exit option (Picture Viewer project), 248
 - referencing, 247
 - reserved words, 250
- constructor methods, 337**
- container objects, forms as, 162**
- container windows, MDI forms, 143**
- Context Menu Strip control, 206-207**
- context menus, 206-207**
- context sensitive help, 56**
- ContextMenuStrip property, Context Menu Strip control, 207**
- Control Box button, adding to forms, 109-110**
- control objects, 60**
- controls**
 - aligning, 132
 - anchoring, 135-138
 - autosizing, 135-138
 - defining, 18
 - forms, 18
 - adding invisible controls to, 21-23
 - adding to via toolbox, 40-42, 124
 - adding visible controls to, 20-21
 - drawing on, 125-140
 - Snap to Lines layout feature, 128
 - Graphics objects, creating, 372
 - grid settings, 126-127
 - groups of
 - selecting, 129-131
 - setting property values in, 133-134
 - layering, 140
 - OpenFileDialog control, 22, 25, 28
 - Picture Viewer project, adding to, 18-23
 - properties, setting in grouped controls, 133-134

controls

- SaveFileDialog control, 22, 25
- sizing, 133
- spacing, 133
- tab order
 - creating, 138-140
 - removing controls from, 140
- Convert class, common conversion methods, 245**
- Convert.ToBoolean() method**
 - Registry object, 420
 - System.IO.File objects, 409
- Convert.ToString() method, Registry object, 420-421**
- Copy() method, System.IO.File objects, 402-403**
- copying files, 402-403
- Count property, SelectedItems collection, 190**
- Create: Project link (Recent Projects category), 32**
- CreateDirectory() method, System.IO.Directory objects, 409**
- CreateGraphics() method, 69-70**
- CreatePrompt property, SaveFileDialog control, 401**
- CreateSubKey() method, Registry object, 417**
- CTR (Common Type System), 484**
- CurrentConfig property, Registry object, 416**
- CurrentUser property, Registry object, 416**
- custom dialog boxes, creating, 357-360
- custom events, object interfaces, 338

- Custom tab (Properties window color drop-down list), 46**
- customizing forms**
 - background colors, 105
 - background images, 106-108
 - borders, 110-112
 - button additions, 109-110
 - icons, 108-109
 - sizing, 112

D

- DashStyle property, Pen objects, 375**
- data encapsulation via classes, 334-335**
- Data Source parameter, ConnectionString property, 439**
- data storage**
 - text files, 413
 - Picture Viewer Project, 429-434
 - reading, 427-429
 - writing, 425-427
 - Windows Registry, 413
 - accessing, 416
 - HKEY_CLASSES_ROOT node, 414
 - HKEY_CURRENT_CONFIG node, 414
 - HKEY_CURRENT_USER node, 414, 417
 - HKEY_LOCAL_MACHINE node, 414, 417
 - HKEY_USERS node, 414
 - Picture Viewer Project, 419-424

- Registry key creation, 416-417
- Registry key deletion, 418
- Registry object, 416
- REG_BINARY data type, 415
- REG_EXPAND_SZ data type, 415
- REG_MULTI_SZ data type, 415
- REG_SZ data type, 415
- retrieving Registry key values, 419
- setting Registry key values, 418
- structure of, 414-415
- using statements, 416
- viewing, 425

data types

- casting, 245
- defining, 242
- determining, 244
- prefixes, 258
- reference types, 243
- signed types, 244
- unsigned types, 243
- value range of, 243
- value types, 243

DataAdapter objects, 438, 441-442**databases**

- ADO.NET connections, 438-440
- data source connections, closing, 440
- DataAdapter objects, 441-442

- DataRow objects
 - Add() method, 448-450
 - columns, 444
 - Delete() method, 450
 - field references in, 444-445
 - ShowCurrentRecord() method, 446-448
 - Update() method, 448
- DataTable objects, 441
- records
 - creating, 448-450
 - deleting, 450
 - editing, 448
 - navigating, 446-448
 - updating, 448
- running, 451
- DataReader object, 438**
- DataRow objects**
 - Add() method, 448-450
 - columns, 444
 - Delete() method, 450
 - field references in, 444-445
 - ShowCurrentRecord() method, 446-448
 - Update() method, 448
- DataSet object, 438**
- DataTable objects, 438, 441**
- DateTime class**
 - AddDays method, 280
 - AddHours method, 280
 - AddMilliseconds method, 280
 - AddMinutes method, 280
 - AddMonths method, 280
 - AddSeconds method, 280
 - AddYears method, 280
 - dates/times, formatting, 282
 - Day property, 281
 - Hour property, 281
 - Minute property, 281
 - Month property, 281
 - Now property, 180, 282
 - parts of dates, retrieving, 281
 - Second property, 281
 - Today property, 282
 - Year property, 281
- DateTime data type, 244**
- DateTime variable, 279**
 - DayOfWeek() property, 281
 - formatting dates/times, 281-282
 - Hour property, 281
 - strings, passing to, 279
- Day property, DateTime class, 281**
- DayOfWeek() property, DateTime variable, 281**
- debugging**
 - code
 - adding comments, 310-312
 - break points, 315
 - build errors, 312, 314
 - catch statements, 323-325
 - finally statements, 323-325
 - Immediate window, 317-320
 - Output window, 321
 - runtime errors, 312-314
 - structured exception handling, 322, 325-329
 - try blocks, 323
 - try statements, 323-325
 - writing error handlers, 323-325
 - Picture Viewer Project, Windows Registry, 422-424
- decimal data type, 244**
- decision statements**
 - else statements, 288-289
 - false expressions, 288
 - if statements, 285-286
 - false expressions, 287
 - nesting, 289
 - switch statements, 290-294
- declaring variables, 249**
- Define Color dialog (Custom tab), 46**
- Delete() method**
 - DataTable objects, 450
 - System.IO.Directory objects, 410
 - System.IO.File objects, 404-405
- DeleteSubKey() method, Registry object, 418**
- DeleteSubKeyTree() method, Registry object, 418**
- deleting**
 - database records, 450
 - event handlers, 89
 - event procedures, 232
 - files, 52-53, 404-405
 - graphics from forms, 383
 - items from lists via code, 190
 - menu items from top-level menus, 202
 - objects, 374
 - procedures, 231-232

deploying applications

deploying applications

- ClickOnce technology, 469-470
 - advanced settings, 475
 - application creation, 471-472
 - Picture Viewer project installation, 474
- uninstalling distributed applications, 474-475

Description section (Properties window), 47

design windows

- closed windows, 35
- displaying, 35
- docking, 35-37
- floating, 35-36
- hiding, 35, 38

destructor methods, 337

dialog boxes

- buttons
 - Accept button, 160
 - Cancel button, 161
- custom dialog boxes, creating, 357-360
- OK button, 159
- tabbed dialog boxes, creating, 181-184

DialogResult property, MessageBox.Show() function, 355-359

Directory flag (file attributes), 406

displaying

- design windows, 35
- object properties, 13

- static text via Label control, 151-153
- toolbars, 39

Dispose() method, 72, 347, 374

distributable components, defining, 8

distributed applications, uninstalling, 474-475

division (/) operator, 269

do...while loops, 303-305

docking

- design windows, 35-37
- toolbars, 40

double data type, 244

double-clicking Visual Studio 2008, 12

drag handles (toolbars), 40

Draw Border button, adding to Picture Viewer project, 68-72

DrawEllipse() method, Graphics object, 381

DrawImage() method, 387

drawing

- circles, 381
- controls on forms, 125
 - aligning controls, 132
 - anchoring controls, 135-138
 - autosizing controls, 135-138
 - grid settings, 126-127
 - grouping controls, 129-131
 - setting grouped control property values, 133-134
- sizing controls, 133

- Snap to Lines layout feature, 128
- spacing controls, 133
- tab order, 138-140
- ellipses, 381
- rectangles, 381

DrawLine() method, Graphics object, 380

DrawRectangle() method, 71, 381

DrawString() method, Graphics object, 382

DRIVER parameter, ConnectionString property, 439

drop-down lists, creating in combo boxes, 172-174

drop-down menus, toolbar buttons, 214

DropDownButton property, ToolStrip control, 214

DropDownList property, combo boxes, 173

DropDownStyle property, combo boxes, 173

dynamism (methods), 68

E

early binding, 344-345

editing database records, 448

ellipses, drawing, 381

else statements

- false expressions, 288
- nesting, 289

Enabled property

- multiline text boxes, 155
- Timer control, 180

- encapsulating data/code via classes, 334-335
- ending programs, 26-27
- endless loops, 303
- Enlarge button, adding to Picture Viewer project, 63, 66
- Environment Tutorial project, 34
 - design windows
 - closed windows, 35
 - displaying, 35
 - docking, 35-37
 - floating, 35-36
 - hiding, 35, 38
 - object properties, 42
 - changing, 43-45
 - color properties, 45-46
 - viewing, 43
 - viewing descriptions of, 47
 - toolbars
 - displaying, 39
 - docking, 40
 - hiding, 39
 - sizing, 40
 - toolbox, adding controls to forms, 40-42
- equal sign (=), setting properties, 61
- erasing code, 65
- error handlers, writing
 - catch statements, 323-325
 - finally statements, 323-325
 - try blocks, 323
 - try statements, 323-325
- Error icon, message boxes, 354
- Error List, 90
- errors
 - build errors, 312-314
 - runtime errors, 312-314
- escape sequences, slashes (\) as, 250
- event handlers
 - creating, 92-95
 - defining, 24
 - deleting, 89
- event-driven programming, 82
- events
 - build example
 - event handler creation, 92-95
 - user interface, 91
 - choosing, 364
 - Click events, 23
 - custom events, object interfaces, 338
 - event handlers
 - creating, 92-95
 - deleting, 89
 - event procedures, 82
 - event-driven programming, 82
 - invoking, 82
 - via objects, 83
 - via OS, 84
 - via user interaction, 83
 - objects, accessing events via, 85-86
 - parameters, 87-88
 - procedures, deleting, 232
 - recursive events, avoiding, 84
- Events button (Properties Window), 86
- Excel
 - ActiveCell objects, 457
 - Application objects, 455-457
 - automation
 - adding cell data, 457-458
 - bold cells, 458
 - creating library references, 454
 - selecting cells, 458
 - server creation, 455-456
 - testing, 459
 - viewing via, 456
 - workbook creation, 457
 - workbooks, creating, 457
 - worksheets
 - adding cell data, 457-458
 - bold cells, 458
 - selecting cells, 458
- exception handling, structured
 - exception handling, 322, 325-329
- Exception objects, Message property, 325
- Exception variables, catch statements, 326
- execution falling through, 294
- Exists() method
 - System.IO.Directory objects, 410
 - System.IO.File objects, 402
- exiting methods, 235
- explicit casting, data types, 245

expressions

expressions

- false expressions
 - else statements, 288
 - if statements, 287
- operator precedence, 270-271
- variables, uses in, 251

F

false expressions

- else statements, 288
- if statements, 287

FileAttributes variable,

GetAttributes() method, 406

FileName property,

OpenFileDialog control, 398

files

- browsing, 24-26
- copying, 402-403
- deleting, 404-405
- log files, Picture Viewer Project, 429-434
- moving, 403-404
- OpenFileDialog control, 396
 - CheckFileExists property, 399
 - FileName property, 398
 - Filter property, 398
 - FilterIndex property, 398
 - InitialDirectory property, 397
 - Multiselect property, 399
 - ShowDialog() method, 399
 - Title property, 398

projects

- adding to, 52-53
- removing from, 52-53

properties, retrieving, 405

- Archive flag, 406
- date/time, 406
- Directory flag, 406
- Hidden flag, 406
- Normal flag, 407
- ReadOnly flag, 407
- System flag, 407
- Temporary flag, 407
- writing code for, 407-409

renaming, 404

SaveFileDialog control, 399

- CreatePrompt property, 401
- OverwritePrompt property, 400

source file existence, determining, 402

System.IO.Directory objects, 401

- CreateDirectory() method, 409
- Delete() method, 410
- Exists() method, 410
- Move() method, 410

System.IO.File objects, 401

- Convert.ToBoolean() method, 409
- Copy() method, 402-403
- Delete() method, 404-405
- Exists() method, 402
- GetAttributes() method, 406, 409
- GetCreationTime() method, 406, 409

GetLastAccessTime()
method, 406, 409

GetLastWriteTime()
method, 406, 409

Move() method, 403-404

SourceFileExists() method,
402

text files, Picture Viewer
Project, 413

displaying log files,
431-433

log file creation, 429-431

testing logs, 433-434

reading, 427-429

writing, 425-427

**Fill method, DataAdapter objects,
441**

Filter property, 23, 398

**FilterIndex property,
OpenFileDialog control, 398**

finally statements, 323-325

float data type, 244

floating design windows, 35-36

Font object, 382

Font property, 44

for loops, 297-302

for statements

components of, 298

for loops, 298-299

form objects, 60

Form_Load events, 442, 445

formatting dates/times, 281-282

FormBorderStyle property, 111

FormClosed events, 365, 388

FormClosing events, 441

forms

BackgroundImage property, 106-109

backgrounds

adding images to, 106-108

changing color, 105

removing images from, 108

borders, customizing, 110-112

buttons

Accept button, 160

adding to, 109-110

Cancel button, 161

OK buttons, 159

check boxes, 161

combo boxes, 166, 172

container objects as, 162

controls, 18

adding invisible controls to, 21-23

adding to via toolbox, 40-42, 124

adding visible controls to, 20-21

drawing on, 125-140

Snap to Lines layout feature, 128

defining, 11-13, 101-102

display position, specifying, 115-116

FormBorderStyle property, 111

graphics, removing, 383

Graphics objects, creating, 372

group boxes, 162-163

hiding, 118

Icon property, 109

icons, adding to, 16-17, 108-109

instantiating, syntax of, 113

list boxes, 166-167

Add() method, 172

adding items to lists, 168

clearing lists, 170

manipulating items at design time, 167

removing items from lists, 169

retrieving item information from lists, 171

Sorted property, 172

MaximumSize property, 112

MDI forms, 143-147

menus

accelerator keys, 200

adding, 198-200

assigning shortcut keys to menu items, 208

checked menu items, 202

context menus, 206-207

creating menu items, 201

creating top-level menus, 198-200

deleting menu items, 202

hotkeys, 200

moving menu items, 202

programming, 203-206

Type Here boxes, 200

MinimumSize property, 112

modality, 114-115

naming, 102

nonmodal forms, 114-115

nonmodal windows, creating
topmost nonmodal windows, 141

panels, 162-163

Picture Viewer project

adding controls, 18-23
sizing, 17

project management, 51

properties, viewing via
Properties window, 103

radio buttons, 164-165

scrollable forms, 142

showing, 113

ShowInTaskbar property, 118

Size.Height property, 171

sizing, 17, 112, 116-117

StartPosition property, 115-116

taskbar, preventing from displaying in, 118

text boxes, adding to, 153

Text property, changing, 15

title bars, displaying text on, 104

toolbars

adding, 209

adding buttons to, 210-212

button drop-down menus, 214

programming, 213-214

transparent forms, creating, 141

Visible property, 113, 118

windows versus, 101

WindowState property, 116-117

FormulaR1C1 property

FormulaR1C1 property, ActiveCell object, 457
frames, 163
FromImage() method, Graphics object, 374
FullRowSelect property, List View control, 189
functions, exposing methods as, 343

G

garbage collection (.NET Framework), 484-485
garbage collector, 337
GDI (Graphical Device Interface), 372
get construct
 read-only properties, creating via, 342
 readable properties, creating via, 341
GetAttributes() method, System.IO.File objects, 406, 409
GetCreationTime() method, System.IO.File objects, 406, 409
GetLastAccessTime() method, System.IO.File objects, 406, 409
GetLastWriteTime() method, System.IO.File objects, 406, 409
GetValue() method, Registry object, 419

graphics

bitmaps, creating for, 373-374
circles, drawing, 381
controls, creating for, 372
ellipses, drawing, 381
forms
 creating for, 372
 removing from, 383
GDI, 372
lines, drawing, 380
pens, 375-376
project example, 383-388
rectangles
 creating, 379
 drawing, 381
 sizing, 380
removing, 374
text as, 382

Graphics objects

bitmaps, creating for, 373-374
Clear() method, 381
controls, creating for, 372
Dispose() method, 374
DrawEllipse() method, 381
DrawLine() method, 380
DrawRectangle() method, 381
DrawString() method, 382
forms, creating for, 372
FromImage() method, 374

grids

controls, 126-127
GridSize property, 126-127
LayoutMode property, 127

ShowGrid property, 127-128
SnapToGrid property, 127-128

Group Box controls, 162-163
grouping controls, 129-131

H

Height property, sizing forms, 17
help

 context sensitive help, 56
 finding, 55-56
 Run mode, 56

Hidden flag (file attributes), 406

Hide() method, 119

hiding

 design windows, 35, 38
 forms, 118
 toolbars, 39

HKEY_CLASSES_ROOT node (Windows Registry), 414

HKEY_CURRENT_CONFIG node (Windows Registry), 414

HKEY_CURRENT_USER node (Windows Registry), 414, 417

HKEY_LOCAL_MACHINE node (Windows Registry), 414, 417

HKEY_USERS node (Windows Registry), 414

hotkeys, 153, 200

Hour property, 281

- I**
 - Icon property, 16, 109**
 - icons**
 - forms, adding to, 16-17, 108-109
 - message boxes
 - displaying in, 353-354
 - Error icon, 354
 - Question icon, 355
 - Picture Viewer project, adding to, 16
 - IDE (Integrated Development Environments)**
 - Properties window, 12
 - displaying object properties in, 13
 - Height property, 17
 - Icon property, 16
 - Name property, 13-15
 - Size property, 17
 - Text property, 15
 - Width property, 17
 - Start page, 9-10
 - Toolbox window, 12
 - Visual Studio 2008 as, 9
 - windows, sizing, 12
- if statements, 285-286**
 - false expressions, 287
 - nesting, 289
- IL (Intermediate Language) code, 481-482**
- Image control, ImageSize property, 185**
- Image List control, 184-185**
- Image property, ToolStrip control, 211**
- ImageIndex property, List View control, 187**
- images, form backgrounds**
 - adding to, 106-108
 - removing from, 108
- ImageSize property, Image control, 185**
- Immediate window, debugging code, 317-320**
- implicit casting, data types, 245**
- IndexOf() method, strings, 277**
- infinite recursion procedures, 237**
- Inflate() method, Rectangle object, 380**
- InitialDirectory property, OpenFileDialog control, 397**
- InitializeComponent() event, 90**
- Insert() method**
 - combo boxes, 172
 - Items collection, 169-170
- instance members, defining, 221**
- instance methods versus static methods, 335**
- instantiating**
 - forms, syntax of, 113
 - objects via classes, 343
 - binding object references to variables, 344-345
 - object creation via variable dimensioning, 346
 - object lifetimes, 347-348
 - releasing object references, 346-347
- int data type, 244**
- int.Parse() method, 287**
- IntelliSense, 64, 88**
- interface design**
 - files, browsing, 24-26
 - terminating programs, 26-27
 - visible controls, adding to forms, 20-23
- interfaces (objects)**
 - client interaction with, 338
 - custom events in, 338
 - defining, 335
 - elements of, 337
 - functions, exposing as methods, 343
 - methods in, 338
 - properties in, 338-340
 - read-only property creation, 342
 - readable property creation via get construct, 341
 - writable property creation via set construct, 341
 - write-only property creation, 342
- Interval property, Timer control, 178**
- Invalidate() method, 388**
- invisible controls, adding to forms, 21-23**
- IsMdiContainer property, 145**
- Items collection**
 - Add() method, 168-170, 189
 - Clear() method, 170, 190
 - Click events, 168-170
 - combo boxes, 172
 - Insert() method, 169-170
 - list boxes, 166
 - adding items to lists, 168
 - clearing lists, 170

Items collection

- manipulating items at design time, 167
- removing items from lists, 169
- retrieving item information from lists, 171
- Remove() method, 169-170, 190
- RemoveAt() method, 169-170
- SelectedIndex method, 171
- SelectedItem method, 171
- ToolStrip control, 210, 213-214
 - DropDownButton property, 214
 - Image property, 211
- Items property, 173, 433**

J - K - L

- jagged arrays, 255**
- JITers (just-in-time compilers), 482**
- keyboards**
 - KeyDown events, 361
 - KeyPress events, 361-363
 - KeyUp events, 361
- KeyChar property, 362**
- Label control**
 - static text, displaying, 151-153
 - TextAlign property, 154
- LargeImageList property, ListView control, 186-188**

- lasso tool, adding control groups to forms, 130-131**
- late binding, 344-345**
- layering controls, 140**
- Layout toolbar**
 - aligning controls, 132
 - Make Horizontal Spacing Equal button, 133
 - Make the Same Size button, 133
 - Save All button, 133
- LayoutMode property, 127**
- Left property, 131**
- Length property, strings, 276**
- libraries, 77**
- lines, drawing, 380**
- List Box control, 166, 191**
- list boxes**
 - Add() method, 172
 - Items collection, 166-167
 - adding items to lists, 168
 - clearing lists, 170
 - manipulating items at design time, 167
 - removing items from lists, 169
 - retrieving item information from lists, 171
 - Location property, 166
 - MultiExtended property, 172
 - MultiSimple property, 172
 - Name property, 166
 - SelectionMode property, 172
 - Size property, 166
 - Sorted property, 172
- List View control, 185, 191**
 - Columns property, 187
 - FullRowSelect property, 189

- ImageIndex property, 187
- Items collection
 - Add() method, 189
 - Clear() method, 190
 - Remove() method, 190
- LargeImageList property, 186-188
- SelectedItem collection, 190
- SubItems property, 188
- Text property, 188
- View property, 188

lists

- adding items to
 - via code, 189
 - via List View, 187-189
- clearing, 190
- clearing items from via code, 190
- columns, creating, 187
- creating, 186
- removing items from via code, 190
- selected items, determining in code, 190

literal values, passing variables to, 250

- Load event, 95, 385**
- local scope, 256-257**
- LocalMachine property, Registry object, 416**
- Location property**
 - buttons, 159
 - Group Box control, 163
 - list boxes, 166
 - radio buttons, 164
 - Tab control, 183

MessageBoxButtons property**log files, Picture Viewer Project**

- creating for, 429-431
- displaying in, 431-433
- testing in, 433-434

logical (Boolean) operators, 273

- And (&) operator, 274
- Not (!) operator, 274
- Or (|) operator, 274
- Xor (^) operator, 275

long data type, 244**loops**

- breaking, 302-303
- do...while loops, 303-305
- endless loops, 303
- for loops, 297-302
- recursive loops, procedures, 237

M**m_cnADONewConnection objects, 442****magic numbers, 246****MainForm Load event, 385****MainForm_FormClosing events, 445****Make Horizontal Spacing Equal button (Layout toolbar), 133****Make the Same Size button (Layout toolbar), 133****managed code, defining, 480****managing projects**

- adding/removing files, 52-53
- class modules, 51
- components of, 50-51
- forms, 51

- setting project properties, 51
- solutions, 50
- user controls, 51
- via Solution Explorer, 48-49

marquee tool, adding control groups to forms, 130**math operators**

- addition (+) operator, 268
- division (/) operator, 269
- expressions, 270-271
- modulus arithmetic, 269
- multiplication (*) operator, 269
- operator precedence, 270-271
- subtraction (-) operator, 269

Maximize button, adding to forms, 109-110**MaximumSize property, 112****MaxLength property, text box characters, 157****MDI (Multiple Document Interface) forms, 143-147****MdiParent property, 145-146****Menu Strip control, 198-206****menus**

- accelerator keys, 200
- context menus, 206-207
- drop-down menus, toolbar buttons, 214
- forms, adding to, 198-200
- hotkeys, 200
- top-level menus

- assigning shortcut keys to menu items, 208
- checked menu items, 202
- creating, 198-200

- creating menu items, 201
- deleting menu items, 202
- moving menu items, 202
- programming, 203-206
- Type Here boxes, 200

message boxes, 351**buttons**

- determining which is clicked, 355-356
- displaying, 353
- displaying, 352
- Error icon, 354
- icons, displaying, 353-354
- message text guidelines, 356-357
- Question icon, 355

Message property, Exception objects, 325**MessageBox.Show() function, 171, 357, 360**

- Buttons property, 352
- Caption property, 352
- DialogResult property, 355-356, 358-359
- MessageBoxButtons property, 352-353
- MessageBoxIcon property, 353-354
- MessageText property, 352
- ShowDialog() method, 359

MessageBox.Show() method, 75**MessageBox.Show() statements, 55****MessageBoxButtons property, MessageBox.Show() function, 352-353**

MessageBoxIcon property

MessageBoxIcon property,

MessageBox.Show() function,
353-354

MessageBox property,

MessageBox.Show() function,
352

method-level scope. **See** local scope

methods, 223

calling, 229-231

constructor methods, 337

declaring

components of, 224

procedures that do not return values, 224-227

procedures that return values, 227-228

destructor methods, 337

dynamism, 68

exiting, 235

exposing functions as, 343

instance methods versus static methods, 335

invoking, 67-68

naming, spaces in, 225

object interfaces, 338

parameters, defining, 225-226

parentheses (), 68

procedures

calling, 229-231

creating, 226

declaring procedures that do not return values, 224-227

declaring procedures that return values, 227-228

deleting, 231-232

infinite recursion, 237

passing parameters, 233-234

recursive loops, 237

properties versus, 68

static methods, 236, 335

Microsoft.VisualBasic name-spaces, 483

Minimize button, adding to forms, 109-110

MinimumSize property, 112

Minute property, DateTime class, 281

modality, forms, 114-115

modulus arithmetic, 269

monitors, system colors

assigning, 378

changing, 376-377

syncing interface colors with user system colors, 377-378

Month property, DateTime class, 281

mouse

click events, 364

MouseDown events, 364

MouseDown events, 86-88, 159, 364

MouseEnter events, 364

MouseHover events, 364

MouseLeave events, 94, 364

MouseMove events, 94, 159, 364-367

MouseUp events, 159, 364

Move() method

System.IO.Directory objects, 410

System.IO.File objects, 403-404

moving

files, 403-404

top-level menu items, 202

multidimensional arrays, 253-254

MultiExtended property, list boxes, 172

Multiline property, 131, 154

multiline text boxes, creating, 154-155

MultilineChanged event, 83

multiplication (*) operator, 269

MultiSelect property

OpenFileDialog control, 399

SelectedItem collection, 190

MultiSimple property, list boxes, 172

N

Name property, 13-15

buttons, 159

control groups, 134

Group Box control, 163

list boxes, 166

radio buttons, 164

namespaces, commonly used

namespaces table, 483-484

naming

forms, 102

methods, spaces in, 225

objects, 13-15

Picture Viewer project, 15

projects, 10

naming conventions

data type prefixes, 258

variable prefixes, 259

navigating database records,
446-448

nesting

- else statements, 289
- if statements, 289

.NET Framework, 480

- CLR, 480-481
- CTR, 484
- garbage collection, 484-485
- IL code, 481-482
- namespaces, 483-484

New Project dialog, 10, 33

Next() method, Random class, 384

nodes, tree view, 77

- adding to, 192-193
- clearing from, 194
- removing from, 194

Nodes collection, 191

- Add() method, 192-193
- Clear() method, 194
- Remove() method, 194

nonmodal forms, 114-115

nonmodal windows, creating, 141

nonstatic methods. *See* instance methods versus static methods

nonvisual controls. *See* invisible controls

Normal flag (file attributes), 407

Not (!) operator, 274

Now property, DateTime class, 180, 282

O

Object Browser, 76

Object data type, 244

objects, 59

binding

- creating objects via variable dimensioning, 346
- references to variables, 344-345

collections, 73-76

control objects, 60

controls

- adding to forms, 18-23
- defining, 18
- OpenFileDialog control, 22, 25, 28
- SaveFileDialog control, 22, 25

defining, 13

events

- accessing, 85-86
- invoking, 83

form objects, 60

forms, instantiating as, 113

garbage collector, 337

instantiating via classes, 343

- binding object references to variables, 344-345

- object creation via variable dimensioning, 346

object lifetimes, 347-348

- releasing object references, 346-347

interfaces

- client interaction with, 338
- custom events in, 338
- elements of, 337
- exposing functions as methods, 343

methods in, 338

properties in, 338-342

libraries. *See* type libraries

lifetime of, 347-348

methods

- dynamism, 68
- invoking, 67-68
- parentheses (), 68
- properties versus, 68

models, 453

naming, 13-15

object-oriented programming, defining, 60

properties

- color properties, 45-46
- defining, 13, 61
- displaying, 13
- Filter property, 23
- Height property, 17
- Icon property, 16
- methods versus, 68
- Name property, 13-15
- Picture Viewer project usage example, 63-66
- read-only properties, 62
- setting, 42-45, 61
- Size property, 17
- syntax of, 61
- Text property, 15
- Title property, 23
- viewing descriptions of, 47
- Width property, 17

Properties window, selecting in, 43

objects

- references, releasing, 346-347
- simple object build example, 68-72
- objFileAttributes** variable, **GetAttributes()** method, 406
- objGraphics()** object, 69-70
- OK** buttons, 159
- OleDbConnection** object, 438
- Opacity** property, 141
- Open File Dialog** control, 178
- OpenFileDialog** control, 22, 25, 28, 395-396
 - CheckFileExists** property, 399
 - FileName** property, 398
 - Filter** property, 398
 - FilterIndex** property, 398
 - InitialDirectory** property, 397
 - Multiselect** property, 399
 - ShowDialog()** method, 399
 - Title** property, 398
- OpenPicture()** function, 429-430
- OpenPicture()** method, 225-226, 236
- operator precedence**, 270-271
- Or (|)** operator, 274
- OS (Operating Systems)**, invoking events, 84
- Output window**, debugging code, 321
- OverwritePrompt** property, **SaveFileDialog** control, 400

P

- Paint** event, 84, 387-388
- Panel** controls, 162-163

- parameters**
 - defining, 54, 226
 - methods, defining in, 225
 - passing between procedures, 233-234
- parameters (events)**, 87-88
- parentheses ()**, methods, 68, 225
- Parse** method, 287
- passing**
 - arguments, 234
 - parameters in procedures, 233-234
- Password** parameter, **ConnectionString** property, 439
- PasswordChar** property, 158
- passwords**, adding to text boxes, 158
- Pen** objects, 375
- pens**, 375-376
- PerformClick** method, 160
- periods (.)**, writing code, 64
- peripherals**
 - keyboards
 - KeyDown** events, 361
 - KeyPress** events, 361-363
 - KeyUp** events, 361
 - monitors
 - assigning system colors, 378
 - changing system colors, 376-377
 - syncing interface colors with user system colors, 377-378
 - mouse
 - click events, 364
 - MouseClick** events, 364

- MouseDown** events, 86-88, 159, 364
- MouseEnter** events, 364
- MouseHover** events, 364
- MouseLeave** events, 94, 364
- MouseMove** events, 94, 159, 364-367
- MouseUp** events, 159, 364

Picture Viewer project

- buttons
 - adding, 63
 - Draw Border** button, 68-72
 - Enlarge** button, 63, 66
 - Show Control Names** button, 74-75
 - Shrink** button, 63, 66
- ClickOnce** install program, 474
- files, browsing, 24-26
- forms
 - adding controls, 18-23
 - sizing, 17
- icons, adding to, 16
- naming, 15
- picture format, selecting, 29
- Prompt on Exit** option, creating constants for, 248
- running, 27-28
- saving, 16
- terminating programs, 26-27
- text files
 - displaying log files, 431-433
 - log file creation, 429-431
 - testing logs, 433-434

- variables
 - creating, 259-260
 - initializing, 261-262
- Windows Registry, 419
 - debugging, 422-424
 - displaying options of, 420-421
 - saving options of, 421
 - stored options of, 421-422
 - testing, 422-424
- pixelformat arguments, 373**
- pixels, defining, 17**
- precedence (operators), 270-271**
- prefixes**
 - data types, 258
 - variables, 259
- private-level scope, 257**
- procedure level scope. See local scope**
- procedures**
 - calling, 229-231
 - creating, 226
 - declaring
 - procedures that do not return values, 224-227
 - procedures that return values, 227-228
 - deleting, 231-232
 - infinite recursion, 237
 - parameters, 54, 233-234
 - recursive loops, 237
 - stacks, 237
 - writing code via, 54-55
- processor independent code. See IL (Intermediate Language) code, 482**
- programming**
 - MessageBox.Show() statements, 55
 - procedures, writing code via, 54-55
 - variables, storing values in, 54
- programs**
 - creating, 11
 - terminating, 26-27
- Project Properties dialog (Solution Explorer), 51**
- projects**
 - creating, 10, 32-33
 - defining, 8
 - existing projects, opening, 34
 - graphics project example, 383-388
 - managing
 - adding/removing files, 52-53
 - class modules, 51
 - components of, 50-51
 - forms, 51
 - setting project properties, 51
 - solutions, 50
 - user controls, 51
 - via Solution Explorer, 48-49
 - naming, 10
 - opening, 9
 - properties, setting, 51
 - running, 27-28
 - saving, 14-16
- Prompt on Exit option (Picture Viewer project), 248**
- properties**
 - controls, setting grouped controls 133-134
 - forms, viewing via Properties window, 103
 - object interfaces, 338-340
 - read-only property creation, 342
 - readable property creation via get construct, 341
 - writable property creation via set construct, 341
 - write-only property creation, 342
 - objects
 - color properties, 45-46
 - defining, 13, 61
 - descriptions, viewing, 47
 - displaying, 13
 - Filter property, 23
 - Height property, 17
 - Icon property, 16
 - methods versus, 68
 - Name property, 13-15
 - Picture Viewer project usage example, 63-66
 - Properties window, 42-45
 - read-only properties, 62
 - setting, 61
 - Size property, 17
 - syntax of, 61
 - Text property, 15
 - Title property, 23
 - Width property, 17
 - projects, setting in, 51
- Properties window, 12**
 - BackColor property, 105
 - color drop-down list, 46

Properties window

Description section, 47
 Events button, 86
 form properties, viewing, 103
 object properties, 42
 changing, 43-45
 color properties, 45-46
 displaying, 13
 Height property, 17
 Icon property, 16
 Name property, 13-15
 Size property, 17
 Text property, 15
 viewing, 43
 viewing descriptions of, 47
 Width property, 17
 Properties pane, setting object properties, 43
 Provider parameter, **ConnectionString** property, 439
 Publish Wizard, **ClickOnce Applications**, 471-472, 475

Q - R

Question icon, message boxes, 355

radio buttons

Checked property, 165
 Location property, 164
 Name property, 164
 Text property, 164
 Random class, **Next()** method, 384

Range objects, 457-458
read-only properties, 62, 342
readable properties, creating via get construct, 341
ReadOnly flag (file attributes), 407
ReadToEnd() method, 428-429, 433
Recent Projects category (Start page)
 Create: Project link, 32
 existing projects, opening, 34
RecordSet object, 438
Rectangle object, 379-380
rectangles, drawing, 381
recursive events, avoiding, 84
recursive loops, procedures, 237
reference data types, 243
reference tracing garbage collection (.NET Framework), 485
Registry (Windows), 413
 accessing, 416
 HKEY_CLASSES_ROOT node, 414
 HKEY_CURRENT_CONFIG node, 414
 HKEY_CURRENT_USER node, 414, 417
 HKEY_LOCAL_MACHINE node, 414, 417
 HKEY_USERS node, 414
 Picture Viewer Project, 419
 debugging, 422-424
 displaying Registry options, 420-421
 saving Registry options, 421

 stored Registry options, 421-422
 testing, 422-424
 Registry keys
 creating, 416-417
 deleting, 418
 retrieving values of, 419
 setting values of, 418
 Registry object
 ClassesRoot property, 416
 Convert.ToBoolean() method, 420
 Convert.ToString() method, 420-421
 CreateSubKey() method, 417
 CurrentConfig property, 416
 CurrentUser property, 416
 DeleteSubKey() method, 418
 DeleteSubKeyTree() method, 418
 GetValue() method, 419
 LocalMachine property, 416
 SetValue() method, 418
 Users property, 416
 REG_BINARY data type, 415
 REG_EXPAND_SZ data type, 415
 REG_SZ data type, 415
 structure of, 414-415
 using statements, 416
 viewing, 425

REG_BINARY data type (Windows Registry), 415

REG_EXPAND_SZ data type (Windows Registry), 415

REG_MULTI_SZ data type (Windows Registry), 415

REG_SZ data type (Windows Registry), 415

Remove() method

Items collection, 169-170

List View, 190

Tree View control, 194

RemoveAt() method, Items collection, 169-170

removing

controls from tab order, 140

database records, 450

files, 52-53, 404-405

graphics from forms, 383

items from lists via code, 190

nodes from tree view, 194

objects, 374

renaming files, 404

Replace() method, strings, 279

reserved words, determining, 250

Resize event, 135

return statements, exiting methods, 235

Run mode, help in, 56

running projects, 27-28

runtime errors, 312-314

S

Save All button (Layout toolbar), 133

SaveFileDialog control, 22, 25, 399

CreatePrompt property, 401

OverwritePrompt property, 400

saving

Picture Viewer project, 16

projects, 14-16, 65

Windows Registry options, Picture Viewer Project, 421

sbrMyStatusStrip control, 430

scope

block scope, 255-256

browsing, 76

defining, 255

local scope, 256-257

private-level scope, 257

variable prefixes, denoting via, 259

scrollable forms, 142

scrollbars, adding to text boxes, 156

ScrollBars property, 156

Second property, DateTime class, 281

Select method, Range objects, 457

SelectedIndex method, Items collection, 171

SelectedIndex property, text boxes, 171

SelectedIndexChanged events, Tab control, 184

SelectedItem method, Items collection, 171

SelectedItem property, SelectedItems collection, 190

SelectedItems collection, ListView control, 190

selecting

multiple controls, 129-131

objects in Properties window, 43

Selection objects, TypeText() method, 462

SelectionMode property, list boxes, 172

SelectNextControl() method, 140

semicolons (;), statements, 65

SendToBack() method, layering controls, 141

separators, 212

SERVER parameter, ConnectionString property, 439

servers, 336

creating instances of, 455-456, 461-463

defining, 453

Excel automation

adding cell data, 457-458

bold cells, 458

selecting cells, 458

server creation, 455-456

testing, 459

viewing, 456

workbook creation, 457

Word automation, server creation, 461-463

set construct, creating writable properties via, 341

SetValue() method

SetValue() method, Registry object, 418

shapes

- circles, drawing, 381
- ellipses, drawing, 381
- rectangles
 - creating, 379
 - drawing, 381
 - sizing, 380

short data type, 244

shortcut keys, assigning to menu items, 208

shortcut menus. See context menus

ShortcutKeys property, 208

Show Control Names button, adding to Picture Viewer project, 74-75

Show() method, 113-115

ShowCurrentRecord() method, DataTable objects, 446-448

ShowDialog() method, 115, 359, 399

ShowGrid property, 127-128

showing forms, 113

ShowInTaskbar property, 118

Shrink button, adding to Picture Viewer project, 63, 66

signed data types, 244

Size property, 45

- forms, sizing, 17
- Group Box control, 163
- list boxes, 166

Size.Height property, 147, 171

Size.Width property, 147

sizing

- controls, 133-138
- forms, 17, 112, 116-117

rectangles, 380

toolbars, 40

windows (IDE), 12

SizingGrip property, Status Bar control, 216

slashes (\) as escape sequences, 250

Snap to Lines layout feature, drawing controls on forms, 128

SnapToGrid property, 127-128

Solution Explorer

managing projects via, 48-49

Project Properties dialog, 51

solutions

- defining, 8
- project management, 50

Sorted property, 172

SourceFileExists() method, System.IO.File objects, 402

spaces

- methods, naming, 225
- strings, trimming from, 278

spacing controls, 133

SqlConnection object, 438

StackOverflow exceptions, 84

stacks, 237

Start page, 9

New Project dialog, 33

New Project page, 10

Recent Projects category

Create: Project link, 32

opening existing projects, 34

starting Visual Studio 2008, 9

StartPosition property, 115-116

statements

block statements, braces
({ }, 286

semicolons (;), 65

static members, defining, 221

static methods, 236, 335

static text, displaying via Label control, 151-153

Status Bar control, 214

SizingGrip property, 216

StatusStrip property, 215

status bars, creating, 214-215

storing data, 413

text files

Picture Viewer Project, 429-434

reading, 427-429

writing, 425-427

Windows Registry

accessing, 416

HKEY_CLASSES_ROOT
node, 414

HKEY_CURRENT_CONFIG
node, 414

HKEY_CURRENT_USER
node, 414, 417

HKEY_LOCAL_MACHINE
node, 414, 417

HKEY_USERS node, 414

Picture Viewer Project, 419-424

Registry key creation, 416-417

Registry key deletion, 418

Registry object, 416

REG_BINARY data type, 415

System.XML namespaces

- REG_EXPAND_SZ data type, 415
- REG_MULTI_SZ data type, 415
- REG_SZ data type, 415
- retrieving Registry key values, 419
- setting Registry key values, 418
- structure of, 414-415
- using statements, 416
- viewing, 425
- StreamReader object**
 - ReadToEnd() method, 428-429
 - text files, reading, 427-429
 - while loops, 429
- StreamWriter object**
 - text files, writing, 425-427
 - Write() method, 426
 - WriteLine() method, 426-427
- strFirstName variable, 54**
- String Collection Editor, adding items to, 167**
- string data type, 244**
- string manipulation**
 - concatenation, 275
 - DateTime variable, passing strings to, 279
 - IndexOf() method, 277
 - Length property, 276
 - Replace() method, 279
 - spaces, trimming, 278
 - String.Remove() method, 278
 - String.Trim() method, 278
 - String.TrimEnd() method, 278
 - String.TrimStart() method, 278
 - Substring() method, 276
 - text, replacing, 278
- String.Remove() method, strings, 278**
- String.Trim() method, strings, 278**
- String.TrimEnd() method, strings, 278**
- String.TrimStart() method, strings, 278**
- StringBuilder variable, 409**
- structure scope. See block scope**
- structured exception handling, 322, 325-326**
 - anticipated exceptions, 326-329
- SubItems property, List View control, 188**
- Substring() method, strings, 276**
- subtraction (-) operator, 269**
- switch statements, 290-294**
- system colors**
 - assigning, 378
 - changing, 376-377
 - syncing interface colors with user system colors, 377-378
- System flag (file attributes), 407**
- System namespaces, 483**
- System palette tab, 377**
- System.Data namespaces, 483**
- System.Diagnostics namespaces, 483**
- System.Drawing namespaces, 483**
- System.IO namespaces, 483**
- System.IO.Directory objects, 401**
 - CreateDirectory() method, 409
 - Delete() method, 410
 - Exists() method, 410
 - Move() method, 410
- System.IO.File objects, 401**
 - Convert.ToBoolean() method, 409
 - Copy() method, 402-403
 - Delete() method, 404-405
 - Exists() method, 402
 - GetAttributes() method, 406, 409
 - GetCreationTime() method, 406, 409
 - GetLastAccessTime() method, 406, 409
 - GetLastWriteTime() method, 406, 409
 - Move() method, 403-404
 - SourceFileExists() method, 402
- System.Net namespaces, 483**
- System.Security namespaces, 483**
- System.Web namespaces, 484**
- System.Windows.Forms namespaces, 484**
- System.XML namespaces, 484**

Tab control

T

Tab control, 177, 182

- Location property, 183
- SelectedIndexChanged events, 184

TabPages property, 181

tab order (controls)

- creating, 138-140
- removing controls from, 140

tabbed dialog boxes, creating, 181-184

TabIndex property, 138-140

TabPages property, Tab control, 181

TabStop property, 140

taskbar

- forms, preventing from displaying in, 118
- ShowInTaskbar property, forms, 118

tbrMainToolBar control, 213

Temporary flag (file attributes), 407

terminating programs, 26-27

testing

- Excel automation, 459
- form modality, 115
- log files, Picture Viewer Project, 433-434
- objects, simple object build example, 72
- Picture Viewer Project, Windows Registry, 422-424

text

- as graphics, 382
- character limits, setting in text boxes, 157

Font property, 44

form title bars, displaying on, 104

static text, displaying via Label control, 151, 153

strings

- concatenation, 275
- replacing within, 278

Text Box control, 153

- Click events, 159
- MaxLength property, 157
- MouseDown events, 159
- MouseMove events, 159
- MouseUp events, 159
- MultiLine property, 154
- PasswordChar property, 158
- ScrollBars property, 156
- TextAlign property, 154
- TextChanged events, 158

text boxes

- character limits, setting, 157
- forms, adding to, 153
- multiline text boxes, creating, 154-155
- password fields, 158
- scrollbars, adding to, 156
- SelectedIndex property, 171

text files, 413

- Picture Viewer Project
 - displaying log files, 431-433
 - log file creation, 429-431
 - testing logs, 433-434
- reading, 427-429
- writing, 425-427

Text property, 145

- buttons, 159
- combo boxes, 173

forms, changing in, 15

Group Box control, 163

labels, 152-153

List View control, 188

multiline text boxes, 154

radio buttons, 164

text boxes, 153

TextAlign property, 154

Textbox control, 83

TextChanged event, 83-84, 88

TextChanged events, 158, 364

this.Close() statements, 27

Tick events, Timer control, 179

time/date. **See** DateTime variable

Timer control, 83

- AutoSize property, 179
- Enabled property, 180
- Interval property, 178
- Tick events, 179

Timer event, 84

title bars (forms), displaying text on, 104

Title property, 23, 398

Today property, DateTime class, 282

ToLongTimeString method, 180

toolbars

- buttons
 - adding to, 210-212
 - drop-down menus, 214
 - separators, 212
- displaying, 39
- docking, 40
- drag handles, 40
- forms, adding to, 209
- hiding, 39

- Layout toolbar
 - aligning controls, 132
 - Make Horizontal Spacing Equal button, 133
 - Make the Same Size button, 133
 - Save All button, 133
 - programming, 213-214
 - sizing, 40
 - Tooltips, 132
 - toolbox, adding controls to forms, 40-42, 124**
 - Toolbox window (IDE), 12**
 - ToolStrip control, Items collection, 209-210, 213**
 - DropDownButton property, 214
 - Image property, 211
 - Tooltips (toolbars), 132**
 - ToolTipText property, 407**
 - top-level menus**
 - creating, 198-200
 - menu items
 - assigning shortcut keys to, 208
 - checked menu items, 202
 - creating, 201
 - deleting, 202
 - moving, 202
 - programming, 203-206
 - topmost nonmodal windows, creating, 141**
 - TopMost property, 141**
 - ToString() method, 93**
 - transparent forms, creating, 141**
 - TransparentColor property, Image List control, 185**
 - tree view, nodes, 77**
 - adding to, 192-193
 - clearing from, 194
 - removing from, 194
 - Tree View control, Nodes collection, 177, 191**
 - Add() method, 192-193
 - Clear() method, 194
 - Remove() method, 194
 - troubleshooting, help**
 - context sensitive help, 56
 - finding, 55-56
 - Run mode, 56
 - true/false values. See check boxes**
 - try blocks, 323**
 - try statements, 323-325**
 - two-dimensional arrays, 253**
 - Type Here boxes, menus, 200**
 - type libraries, creating references to**
 - Excel, 454
 - Word, 460
 - TypeText() method, Selection objects, 462**
- ## U - V
- unmanaged code, defining, 480**
 - unsigned data types, 243**
 - Update() method**
 - DataAdapter objects, 441
 - DataTable objects, 448
 - updates, database records, 448**
 - user controls, project management, 51**
 - User ID parameter, ConnectionString property, 439**
 - User Name Label control, 183**
 - Users property, Registry object, 416**
 - using statements, 374**
 - automation server instances, creating, 456
 - structured exception handling, 323
 - Windows Registry, 416
 - value data types, 243**
 - variables**
 - arrays
 - declaring, 252
 - defining, 251
 - dimensions of, 254
 - jagged arrays, 255
 - multidimensional arrays, 253-254
 - referencing variables, 252
 - two-dimensional arrays, 253
 - binding object references to
 - early binding, 345
 - late binding, 344-345
 - creating, 251
 - declaring, 249
 - defining, 62, 241
 - expressions, uses in, 251
 - literal values, passing to, 250
 - object creation via variable dimensioning, 346
 - Picture Viewer project
 - creating for, 259-260
 - initializing in, 261-262

variables

prefixes, denoting scope via,
259

reserved words, 250

storing values in, 54

View property, List View control,
188

visible controls, adding to forms,
20-21

Visible property, 113, 118

Visual Studio 2008 as IDE, 9-12

W

Web tab (Properties window color
drop-down list), 46

while loops, StreamReader
objects, 429

Width property, sizing forms, 17

windows

forms versus, 101

nonmodal windows, 141

sizing, 12

Windows Registry, 413

accessing, 416

HKEY_CLASSES_ROOT node,
414

HKEY_CURRENT_CONFIG
node, 414

HKEY_CURRENT_USER node,
414, 417

HKEY_LOCAL_MACHINE node,
414, 417

HKEY_USERS node, 414

Picture Viewer Project, 419

debugging, 422-424

displaying Registry
options, 420-421

saving Registry options,
421

stored Registry options,
421-422

testing, 422-424

Registry keys

creating, 416-417

deleting, 418

retrieving values of, 419

setting values of, 418

Registry object, 416

Convert.ToBoolean()
method, 420

Convert.ToString() method,
420-421

CreateSubKey() method,
417

DeleteSubKey() method,
418

DeleteSubKeyTree()
method, 418

GetValue() method, 419

SetValue() method, 418

REG_BINARY data type, 415

REG_EXPAND_SZ data type,
415

REG_MULTI_SZ data type,
415

REG_SZ data type, 415

structure of, 414-415

using statements, 416

viewing, 425

WindowState property, 116-117

Word, automation

library references, 460

server creation, 461-463

workbooks (Excel), 457

worksheets (Excel), cells

adding data, 457-458

bold cells, 458

selecting, 458

writable properties, creating via
set construct, 341

Write() method, StreamWriter
object, 426

write-only properties, creating,
342

WriteLine() method, 321,
426-427

writing text files, 425-427

X - Y - Z

Xor (^) operator, 275

Year property, DateTime class,
281

yes/no values. See check boxes

z-order, layering controls, 140