

James Foxall

Sams **Teach Yourself**

# Visual Basic® 2010

in **24**  
**Hours**

SAMS



## **Sams Teach Yourself Visual Basic 2010 in 24 Hours Complete Starter Kit**

Copyright © 2010 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.

ISBN-13: 978-0-672-33113-8

ISBN-10: 0-672-33113-6

Library of Congress Cataloging-in-Publication Data:

Foxall, James D.

Sams teach yourself Visual Basic 2010 in 24 hours complete : starter kit / James Foxall.  
p. cm.

Includes index.

ISBN 978-0-672-33113-8

1. Microsoft Visual BASIC. 2. BASIC (Computer program language) 3. Microsoft .NET. I. Title.  
QA76.73.B3F69529 2010  
006.7'882-dc22

2010011612

Printed in the United States of America

First Printing May 2010

### **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.

### **Warning and Disclaimer**

Every effort has been made to make this book as complete and as 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 DVD 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 of the U.S., please contact

**International Sales**

**international@pearsoned.com**

**Editor-in-Chief**

Karen Gettman

**Executive Editor**

Neil Rowe

**Development Editor**

Mark Renfrow

**Managing Editor**

Patrick Kanouse

**Project Editor**

Mandie Frank

**Copy Editor**

Margo Catts

**Indexer**

Ken Johnson

**Proofreader**

Leslie Joseph

**Technical Editor**

J. Boyd Nolan

**Publishing**

**Coordinator**

Cindy Teeters

**Multimedia**

**Developer**

Dan Scherf

**Designer**

Gary Adair

**Composition**

Mark Shirar

# Introduction

Visual Basic 2010 is Microsoft's latest incarnation of the enormously popular Visual Basic language, and it's fundamentally different from the versions that came before it. Visual Basic is more powerful and more capable than ever before, and its features and functionality are on par with "higher-level" languages such as C++. One consequence of this newfound power is added complexity. Gone are the days when you could sit down with Visual Basic and the online Help and teach yourself what you needed to know to create a functional program.

## Audience and Organization

This book is targeted toward those who have little or no programming experience or who might be picking up Visual Basic 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 large commercial applications with Visual Basic and teaching Visual Basic 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 to a development project. Feel free to post your suggestions or success stories at [www.jamesfoxall.com/forums](http://www.jamesfoxall.com/forums).

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

- ▶ Part I, "The Visual Basic 2010 Environment," teaches you about the Visual Basic environment, including how to navigate and access Visual Basic'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 Basic 2010 programming—and there's a lot to learn. You'll dis-

cover how to create modules and procedures, as well as how to store data, perform loops, and make 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. 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 Basic 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 in an *italic* typeface 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**.

Menu options are separated by a comma. For example, when you should open the File menu and choose the New Project menu option, the text says “Select File, New Project.”

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 using many of Visual Basic's tools to create, debug, and deploy modest Visual Basic programs. 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 Basic mastery.

I love programming with Visual Basic, and sometimes I find it hard to believe I get paid to do so. I hope you find Visual Basic as enjoyable as I do!

## HOUR 1

# Jumping in with Both Feet: A Visual Basic 2010 Programming Tour

---

### ***What You'll Learn in This Hour:***

- ▶ Building a simple (yet functional) Visual Basic 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 2010 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, of course, is one step at a time. I believe the first step to mastering a programming language is *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, and you are free to focus on the mechanics necessary to produce the desired result.

Producing large, commercial solutions is accomplished by way of a series of small steps. After you've finished 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 build on the examples in this hour in subsequent chapters. By the time you complete this book, you will have built a robust application, complete with resizable screens, an intuitive interface including menus and toolbars, manipulation of the Windows Registry, and robust code with professional error handling. But I'm getting ahead of myself.

In this hour, you'll complete a quick tour of Visual Basic that takes you step by step through creating a complete, albeit small, Visual Basic program. Most introductory programming books start by having the reader create 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—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 using Visual Basic 2010.

## Starting Visual Basic 2010

Before you begin creating programs in Visual Basic 2010, 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 Basic 2010 development environment (the tools you use to create a .NET program) to run (although they do require the .NET runtime, which I'll 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 Windows application projects and support dynamic link library (DLL) projects.
- ▶ **Solution:** A collection of projects and files that make up an application or component.

### By the Way

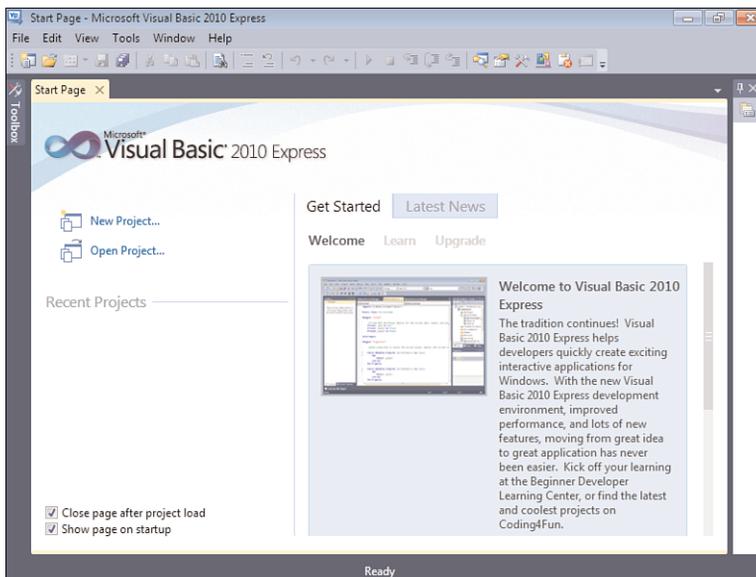
In the past, Visual Basic was an autonomous language. This has changed. Now, Visual Basic is part of a larger entity known as the *.NET Framework*. The .NET Framework encompasses all the .NET technology, including Visual Studio .NET (the suite of development tools) and the common language runtime (CLR), 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 Basic is one of many languages that exist within the Visual Studio family. Many other languages, such as C#, are also .NET languages, make use of the CLR, and are developed within Visual Studio.

Visual Studio 2010 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 Basic projects is accessed from within the Visual Basic IDE. Again, Visual Studio 2010 supports development using many different languages, Visual Basic being the most popular. The environment itself is not Visual Basic, but the language you'll be using within Visual Studio 2010 *is* Visual Basic. To work with Visual Basic projects, you first start the Visual Studio 2010 IDE.

Start Visual Studio 2010 now by choosing Microsoft Visual Basic 2010 Express Edition from the Start/Programs menu. If you are running the full retail version of Visual Studio, your shortcut may have a different name. In this case, locate the shortcut on the Start menu and click it once to start the Visual Studio 2010 IDE.

## Creating a New Project

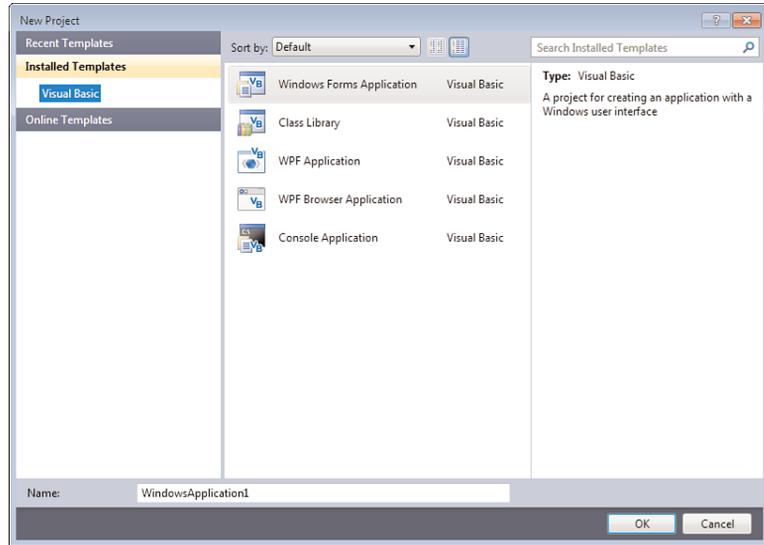
When you first start Visual Studio 2010, you see the Start Page tab within the IDE, as shown in Figure 1.1. You can open projects created previously or create new projects from this Start page. For this quick tour, you'll create a new Windows application, so select File, New Project to display the New Project dialog box shown in Figure 1.2.



**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.



### By the Way

If your Start page doesn't look like the one shown in Figure 1.1, chances are that you've changed the default settings. In Hour 2, "Navigating Visual Basic 2010," I'll show you how to change them back.

The New Project dialog box is used to specify the type of Visual Basic project to create. (You can create many types of projects with Visual Basic, 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 Basic for all examples in this book. If you are running the full version of Visual Studio, you will have many more options available.

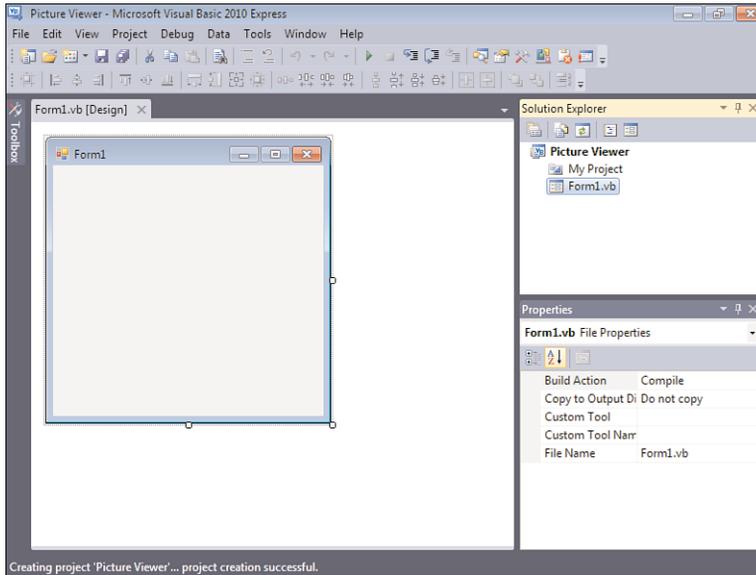
Create a new Windows Forms Application now by following these steps:

1. Make sure that the Windows Forms Application item 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 Basic creates a new Windows Forms Application project, it adds one form (the empty gray window) for you to begin building the *interface* for your application, as shown in Figure 1.3.



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

Within Visual Studio 2010, *form* is the term given to the design-time view of a window that can be displayed to a user.

**By the  
Way**

Your Visual Studio 2010 environment might look different from that shown in the figures in this hour, depending on the edition of Visual Studio 2010 you're using, whether you've already played with Visual Studio 2010, and other factors, such as your monitor's resolution. All the elements discussed in this hour exist in all editions of Visual Studio 2010, however. (If a window shown in a figure doesn't appear 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. 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, that person is 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 2010 Environment

The first time you run Visual Studio 2010, you'll notice that the IDE contains a number of windows, such as the Properties window on the right, which is used to view and set properties of objects. 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 another window. To close the window, 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 close automatically. Instead, it stays open until you click 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 2010 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 appears. 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 on 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 were captured on Windows 7 running at 800×600 because of size constraints, but you should run at a higher resolution if you can. I highly recommend that you develop applications with Visual Basic at a screen resolution of 1024×768 or higher to have plenty of work space. To change your display settings, right-click the desktop and select Screen Resolution. Keep in mind, however, that end users might be running at a lower resolution than you are using for development.

## Changing the Characteristics of Objects

Almost everything you work with in Visual Basic 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. 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 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 will be covered in more detail later. I try to keep forward references to a minimum, but unfortunately, teaching programming is 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 will be covered in greater detail later.

**Watch  
Out!**

Every object has a distinct set of attributes known as *properties* (regardless of whether the object has a physical appearance). Properties define an object’s characteristics. You have certain properties, such as your height and hair color. Visual Basic objects have properties as well, such as Height and BackColor. 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).

Click anywhere in the default form now, and check to see that 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 object’s name, and  
System.Windows.Forms.Form is the object’s type.

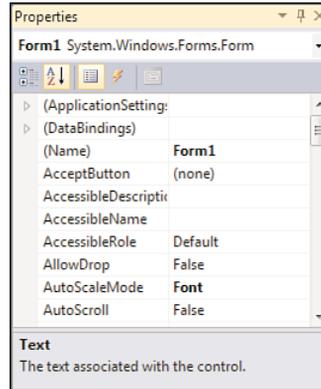
## Naming Objects

The property you should always set first when creating any new object is the Name property. Press F4 to display the Properties window (if it’s not already visible), and scroll toward the top of the properties list until you see the (Name) property, as shown in Figure 1.4. If the Name property isn’t one of the first properties listed, the 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.

**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**

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 called (Name). If you're wondering why it has parentheses around it, it's because the parentheses force the property to the top of the list; 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 within the project, Visual Basic 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 Basic named your form Form1, but it's common to have dozens (or even hundreds) of forms in a project. It would be extremely difficult to manage such a project if all forms were distinguishable only by a number (Form2, Form3, and so forth).

### **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 purposes of this quick tour, I simply call it a form. See Hour 5, "Building Forms: The Basics," for more information.

To better manage your forms, give each one a descriptive name. Visual Basic gives you the chance to name new forms as they're created in a project. Visual Basic created this default form for you, so you didn't get a chance to name it. It's important

not only to change the form's name but also to change its filename. Change the programmable name and the filename 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.vb 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.vb to ViewerForm.vb.

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

**By the  
Way**

The form's Name property is actually changed for you automatically when you rename the file. In future examples, I will have you rename the form file so that the Name property is changed automatically. I had you set it in the Properties window here so that you could see how the Properties window works.

## Setting the Form's Text Property

Notice that the text that appears in the form's title bar says Form1. Visual Basic 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 name of the form. The text in the title bar is determined by the value of the form's Text property. 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 a different property. You'll see the text in the form's title bar 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.

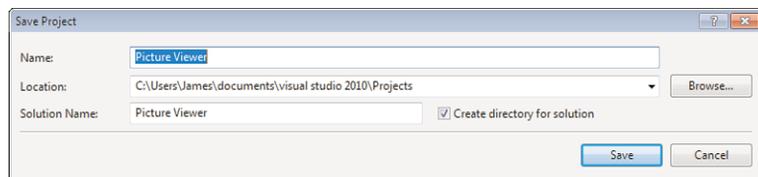
Click the Save All button on the toolbar (the picture of a stack of floppy disks) now to save your work. Visual Basic displays the Save Project dialog box, shown in Figure 1.5. Notice that the Name property is already filled in because you named the project when you created it. The Location text box is where you specify the location in which the project is to be saved. Visual Basic 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 Basic create a solution folder, and if you do Visual Basic creates the solution file in the folder, and it creates a subfolder for the project and the actual files. 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.

## Giving the Form an Icon

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

**FIGURE 1.5**

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



### **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 <http://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'll find the icon for this example in the folder Hour 01\Picture Viewer. You don't have to use the icon I've provided for this example; you can use any icon. 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:

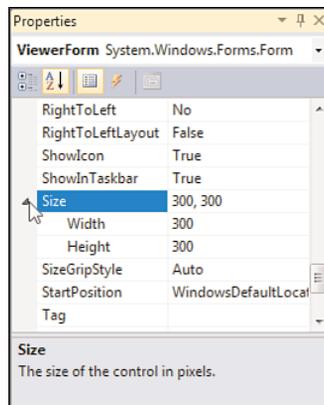
1. In the Properties window, click the Icon property to select it.
2. When you click the Icon property, a small button with three dots appears to the right of the property. Click this button.

- Use the Open dialog box that appears to locate the Picture Viewer.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 choose 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 Form's Size

Next, you'll change the form's Width and Height properties. The Width and Height values are shown collectively under the Size property; Width appears to the left of the comma, and 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 triangle next to the Size property (see Figure 1.6). (After you click it, it changes to a triangle pointing diagonally down.)



**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.

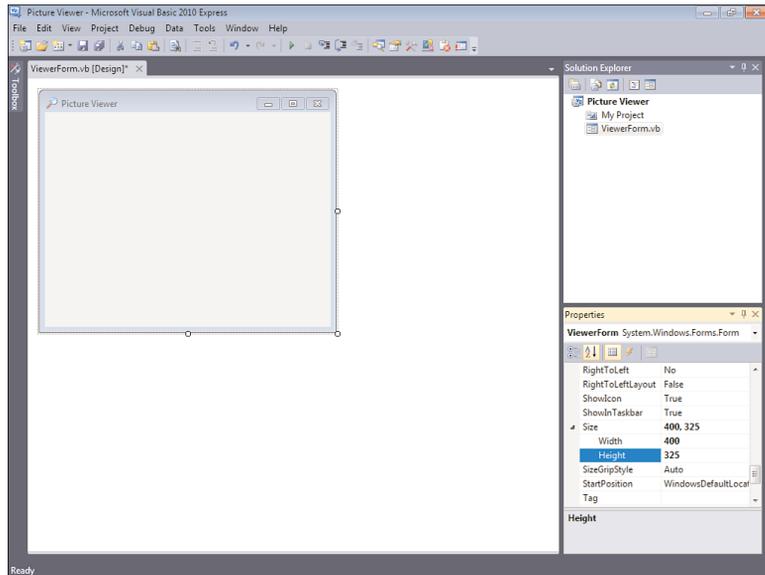
**By the Way**

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 shown in Figure 1.7.

**FIGURE 1.7**

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



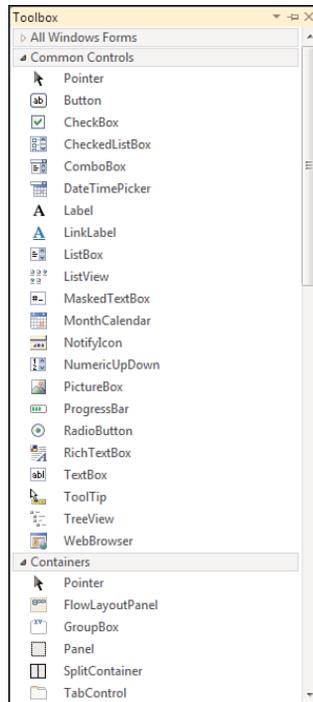
### By the Way

You can also size a form by dragging its border, which you'll learn about in Hour 2, or by using code to change its properties, which you'll learn how to do 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 floppy disks.

## Adding Controls to a Form

Now that you've set the initial properties of your form, 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 to display the Toolbox window to see the most commonly used controls, expanding the Common Controls section if necessary (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

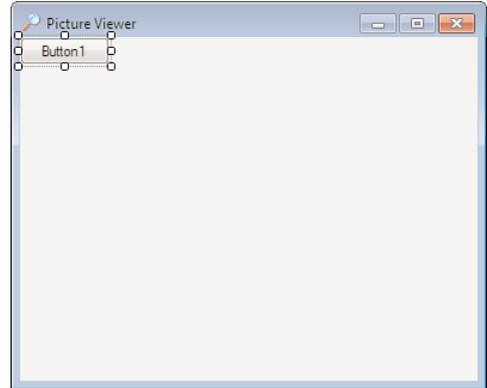
It's generally best to design a form's user interface and then add the code behind the interface to make the form functional. You'll build your interface in the following sections.

## 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 Basic creates a new button and places it in the upper-left corner of the form, as shown in 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 shown in the following list. 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 a while.

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

Now you'll 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 context menu. Next, right-click anywhere on the form, and choose Paste from the form's

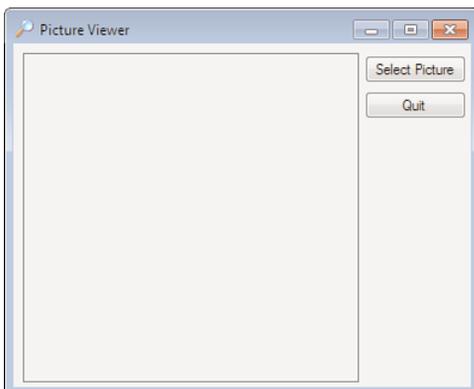
shortcut menu. (You can also use the keyboard shortcuts Ctrl+C to copy and Ctrl+V to paste.) The new button appears centered on the form, and it's selected by default. Notice that it retains almost all the properties of the original button, but the name has been reset. Change the properties of the new button 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 shown 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 you've used so far sit on a form and have a physical appearance when a user runs the application. Not all controls have a physical appearance, however. Such controls, called *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 Basic.

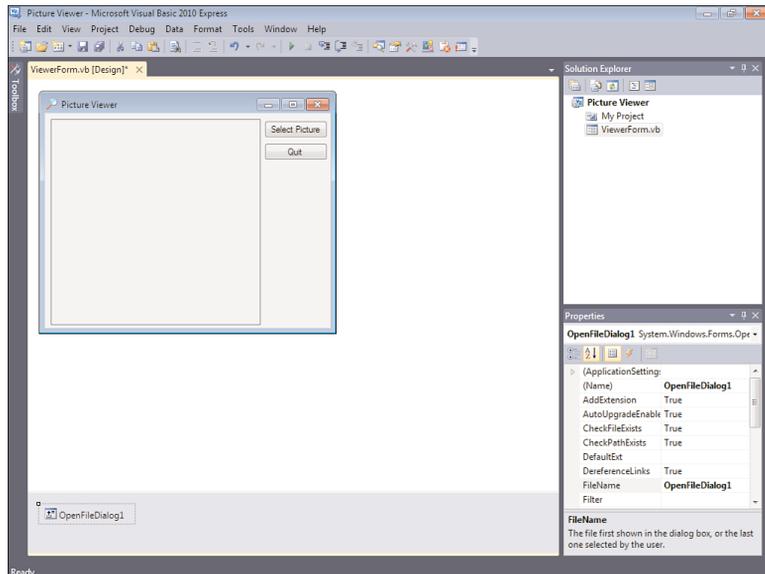
To enable users to select a picture to display, you need to give them the ability to locate a file on their hard drives. You might have noticed that whenever you choose to open a file from within any Windows application, the dialog box displayed is almost always the same. It doesn't make sense to force 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 `OpenFileDialog`, and it will save you 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 allowing the user to specify a filename and path for saving a file.

Display the toolbox 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; rather, it appears in a special area below the form (see Figure 1.11). This

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



happens because the `OpenFileDialog` control has no form interface to display to the 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.

Select the `OpenFileDialog` control and change its properties as follows:

Property	Value
Name	<code>ofdSelectPicture</code>
Filename	<code>&lt;make empty&gt;</code>
Filter	<code>Windows Bitmaps *.BMP JPEG Files *.JPG</code>
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. Click Save All on the toolbar now to save your work.

## 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 Basic 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 anything. 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'll write code to accomplish two tasks. First, you'll write code that lets users browse their hard drives to locate and select a picture file and then display it in the picture box (this sounds a lot harder than it is). Second, you'll add code to the Quit button that shuts down the program when the user clicks the button.

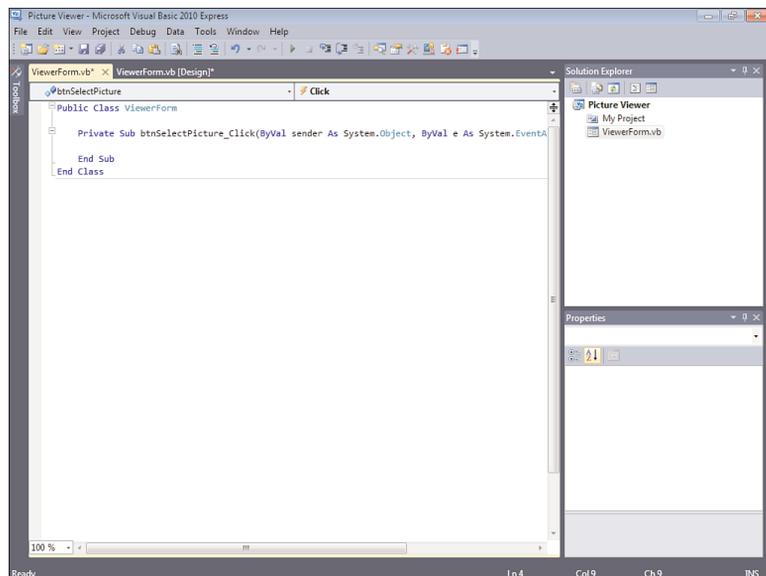
## Letting a User Browse for a File

The first bit of code you'll 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 your code in a window such as this.



When you access an event, Visual Basic 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, 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'll enter is a comment. Beginning a statement with an apostrophe (') designates that statement as a comment. The compiler won't do anything with the statement, so you can enter whatever text you want after the apostrophe. 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'll enter triggers a method of the `OpenFileDialog` control that you added to the form. 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. 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 statement and press Enter to commit the code (don't worry about capitalization; Visual Basic will fix the case for you):

```
If ofdSelectpicture.ShowDialog = DialogResult.OK Then
```

After you insert the statement that begins with `If` and press Enter, Visual Basic automatically creates the `End If` statement for you. If you type in `End If`, you'll wind up with two `End If` statements, and your code won't run. If this happens, delete one of the statements. Hour 13, "Making Decisions in Visual Basic Code," has all the details on the `If` statement.

**By the  
Way**

It's time for another comment. The cursor is currently between the statement that starts with `If` and the `End If` statement. Leave the cursor there and type the following statement, remembering to press Enter at the end of the line:

```
' Load the picture into the picture box.
```

Don't worry about indenting the code by pressing the Tab key or using spaces. Visual Basic automatically indents code for you.

**Did you  
Know?**

This next statement, which appears within the If construct (between the If and End If statements), is the line of code that actually displays the picture in the picture box.

Enter the following statement:

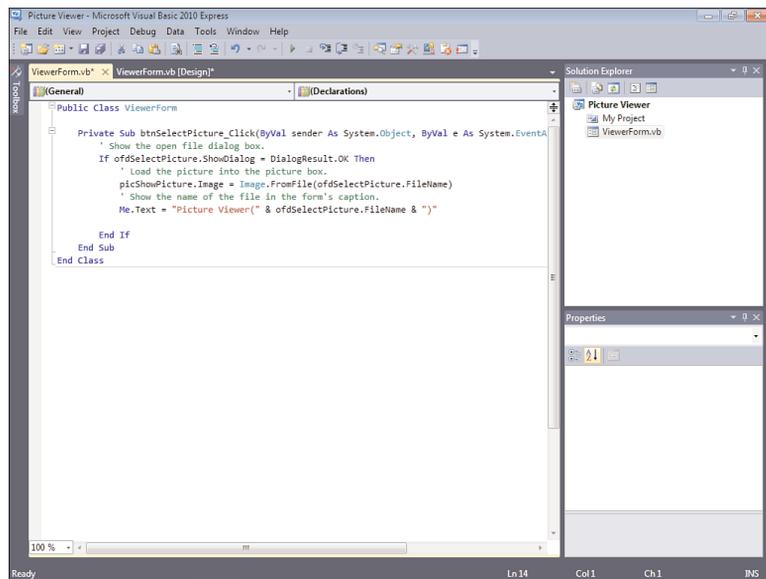
```
picShowPicture.Image = Image.FromFile(ofdSelectPicture.FileName)
```

In addition to displaying the selected picture, your program also displays the path and filename of the picture in the title bar. When you first created the form, you changed its Text property in the Properties window. To create dynamic applications, properties need to be constantly adjusted at runtime, and you do this using code. Insert the following two statements, pressing Enter at the end of each line:

```
' Show the name of the file in the form's caption.
Me.Text = "Picture Viewer(" & ofdselectpicture.FileName & ")"
```

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 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 need to access the Click event handler of the btnQuit button. At the top of the code window are two tabs. The current tab says ViewerForm.vb\*. This tab contains the code window for the form that has the filename ViewerForm.vb. Next to this is a tab that says ViewerForm.vb [Design]\*. Click this tab 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; press Enter at the end of each statement:

```
' Close the window and exit the application  
Me.Close()
```

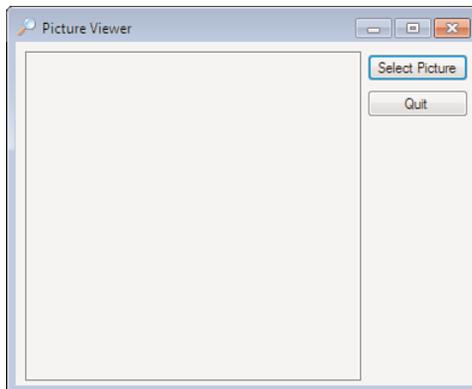
The `Me.Close()` statement closes the current form. When the last loaded form in a program is closed, the application shuts itself down—completely. As you build more robust applications, you'll probably want to execute all kinds of cleanup routines before terminating an application, but for this example, 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 (the stack of floppy 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 player. (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 that you use them whenever possible.

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



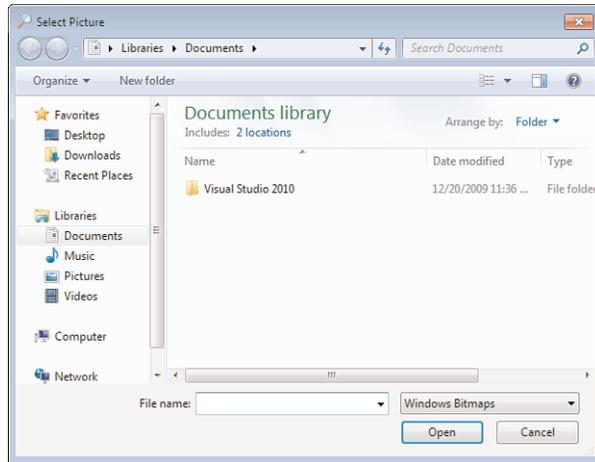
**FIGURE 1.14**  
When in Run mode, your program executes just as it would for an end user.

You are now running your program as though it were a stand-alone application running on another user's machine; what you see is exactly what users would see if they

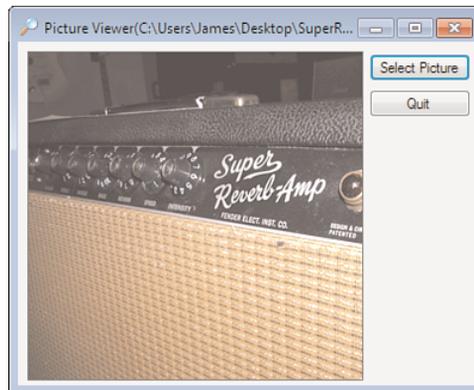
ran the program (without the Visual Studio 2010 design environment in the background, of course). Click the Select Picture button to display the Select Picture dialog box, shown in Figure 1.15. Use this dialog box to locate a picture file. When you've found a file, double-click it, or click once to select it and then click Open. The selected picture is then displayed in the picture box, as shown in Figure 1.16.

**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?



**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 shown in Figure 1.15.

If you want to select and display a picture from your digital camera, chances are the format is JPEG, so you'll 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 Basic 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 Basic. Even the most complicated programs are built using this fundamental approach: 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 Basic (Visual Studio .NET) environment. I then move on to building an interface, and later I teach you about writing code. This organization is deliberate. You might be eager 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 Basic. As you progress through the hours, you'll build 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 bitmaps of file types other than BMP and JPG?*

**A.** Yes. PictureBox supports the display of images with the extensions BMP, JPG, ICO, EMF, WMF, and GIF. PictureBox 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 allow 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

### Quiz

1. What type of Visual Basic 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 the default event (code) of a control?
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

## A

accelerator keys, 198

accept buttons, 154-156

AcceptButton property, 155-156

AcceptsReturn property, 150

ActiveCaption system color, 394

ActiveCaptionText system  
color, 394

ActiveCell objects, 474

ActiveX controls. *See* user con-  
trols

Add() method, 73

addition operations, 270

### ADO.NET

ConnectionString property,  
454-455

DataAdapter objects, 452,  
456-457

databases

closing data source con-  
nections, 455

connecting to, 453-455

DataRow objects, 459-460

DataSet objects, 452

DataTable objects, 452, 456

creating, 458

creating records, 463-464

deleting records, 464

editing records, 462

navigating records,  
460-462

populating, 458

referencing fields in  
DataRows, 459-460

SqlConnection objects,  
452-453

SqlDataAdapter objects,  
456-457

System.Data namespace, 452

### aligning

controls, 125-126

text in text boxes, 148

anchoring controls, 128-130

And operator, 276

### applications

automating

Excel (MS), 470-476

Word (MS), 475-477

## applications

- ClickOnce technology, 481-482
  - distributed applications, uninstalling, 486-487
  - publishing, 482-484, 488
  - testing, 486
  - arithmetic operations**
    - addition, 270
    - comparison operators, 273-274
    - division, 271
    - exponentiation, 271
    - modulus arithmetic, 271
    - multiplication, 271
    - negation, 270
    - order of operator precedence, determining, 272-273
    - subtraction, 270
  - arrays**
    - declaring, 251
    - defining, 237
    - dimensioning, 251
    - multidimensional arrays, 252-254
    - variables, referencing, 251-252
  - attribute flags (files), 421**
  - Auto Hide property, 36**
  - auto-hiding design windows, 36**
  - automation**
    - defining, 469
    - Excel (MS)
      - cell manipulation in workbooks, 473-475
      - forcing showing of Excel, 473
      - instances of automation server creation, 472
      - library reference creation, 470-472
      - testing applications, 475-476
      - workbook creation, 473
    - system requirements, 478
    - Word (MS), 475
      - instances of automation server creation, 477
      - library reference creation, 476-477
  - autosizing controls, 128-130**
- ## B
- BackColor property, 41, 99-100**
  - BackgroundImage property, 100-102**
  - backgrounds (forms)**
    - adding images to, 100-101
    - changing color, 98-99
  - BaseDirectory() method, 444**
  - binding object references to variables, 357**
    - creating new objects, 360-361
    - early binding, 360
    - late binding, 358-359
  - bitmaps, Graphics objects, 391**
  - block (structure) scope, 254-255**
  - Boolean data type, 239-240**
  - Boolean logic**
    - And operator, 276
    - defining, 269, 274-275
    - If...Then constructs, 294
    - Not operator, 276
    - Or operator, 276-277
    - Xor operator, 276-277
  - borders (forms), customizing, 105-107**
  - BorderStyle property, 41**
  - break points, 329-331**
  - browsing files, 22-24**
  - build errors, 326-328**
  - buttons, 81**
    - accept buttons, 154-156
    - Buttons collection, 208-210
    - cancel buttons, 154-156
    - check boxes, 156-157
    - creating, 154
    - radio buttons, 159-160
    - toolbar buttons, 212
    - user messages
      - determining which button is clicked, 372
      - specifying buttons in, 369-371
  - Byte data type, 239**
- ## C
- cancel buttons, 154-156**
  - CancelButton property, 156**
  - Case Else statements, 302**
  - Case statements, 299-302**
  - casting data between data types, 241-242**
  - cells (Excel), 473-475**
  - Char data type, 239**
  - check boxes, 156-157**

- checked menu items, creating, 200-201
  - CheckState property, 156
  - circles, drawing, 398
  - class modules, 48, 218
  - classes
    - defining, 348
    - encapsulating code/data, 348
    - namespaces, 494-495
    - objects, creating, 350-351
      - adding properties to classes, 352-356
      - exposing object attributes as properties, 352-356
    - objects, instantiating
      - binding references to variables, 357-361
      - releasing object references, 362
    - standard modules versus, 349
  - Clear() method, 69, 398
  - clearing
    - lists (list boxes), 165-166
    - nodes from hierarchical lists, 190
  - Click events, 24-25, 81, 84, 153-155, 382, 477
  - ClickOnce technology, 481-482
    - Publish Wizard, 482-484, 488
  - clients, defining, 469
  - Clng() function, 332-334
  - Close() method, 455
  - closing
    - For...Next loops, 310
    - forms, 112-113
    - windows, 10
  - CLR (Common Language Runtime), 492-493
  - code
    - comments, adding, 324-325
    - debugging, 323-324
      - adding comments to code, 324-325
    - break points, 329-331
    - build errors, 326-328
    - creating error handlers via Try...Catch...Finally structures, 336-339
    - handling exceptions in error handlers, 339-344
    - Immediate window, 331-335
    - On Error statements, 337
    - runtime errors, 326-328
  - encapsulating via classes, 348
  - labels, 304
  - modules
    - class modules, 218
    - defining, 217
  - writing, 21
    - browsing files example, 22-24
    - terminating programs example, 24-25
- code procedures
  - calling, 225-230
  - creating, 219-220
    - declaring procedures with return values, 224-225
    - declaring procedures without return values, 220-224
- CLR (Common Language Runtime), 492-493
  - exiting, 231-232
  - infinite recursion, 232
- collections, 73-74. *See also* objects
  - Buttons collection, 208-210
  - Controls collections, 73-75
  - iterative processing, 72
- color
  - background color, changing, 98-99
  - color properties, 42-43
  - dithered colors, 44
  - system colors, 393-396
- columns
  - DataRows, 459
  - enhanced lists, creating columns for, 183
- COM components, 472
- combo boxes, 161, 168-170
- CommandBuilder objects, 457
- comments, adding to code, 324-325
- common language runtime (CLR), 492-493
- common type systems, 496
- comparison operators, 273-274
- compilers, defining, 238
- concatenation, 90, 278-279
- ConnectionString property, 454-455
- constants, 244
  - benefits of, 242-243
  - defining, 237, 242-243
  - naming, 246
  - referencing, 243
  - syntax of, 243

## containers

- containers, 157-158
- context menus, 204-206
- context-sensitive help, 53
- Continue Do statements, 317
- Continue For statements, 312
- control box buttons, adding to forms, 103-104
- control objects, 58
- Control system color, 394
- ControlDark system color, 394
- ControlLight system color, 395
- controls, 145
  - aligning, 125-126
  - anchoring, 128-130
  - autosizing, 128-130
  - buttons
    - accept buttons, 154-156
    - cancel buttons, 154-156
    - check boxes, 156-157
    - creating, 154
    - radio buttons, 159-160
  - check boxes, 156-157
  - combo boxes, 161, 168-170
  - containers, 157-158
  - Context Menu Strip control, 205
  - drawing, 119
  - enhanced lists, creating, 182
    - adding list items, 183-186
    - columns, 183
    - determining selected list items, 186
    - removing list items, 186-187
  - forms, adding to, 16-18, 20-21, 38-39
  - double-clicking controls in toolbox, 118
  - dragging controls from toolbox, 118
  - drawing controls, 119
  - forms, organizing on
    - aligning controls, 125-126
    - anchoring controls, 128-130
    - autosizing controls, 128-130
    - creating tab orders, 131-133
    - grid (size and snap), 120-121
    - layering controls, 133
    - selecting groups of controls, 123-125
    - setting property values for groups of controls, 127
    - sizing controls, 126
    - Snap to Lines, 122-123
    - spacing groups of controls, 126
  - Graphics objects, 390
  - Group Box control, 157-158
  - group boxes, 157-158
  - groups of controls
    - setting property values, 127
    - spacing, 126
  - hierarchical lists, creating, 187
    - adding nodes, 188-189
    - clearing all nodes, 190
    - removing nodes, 190
  - Image List control, 180-181
  - invisible-at-runtime controls, 20-21
  - Label control, 145-147
  - layering, 133
  - list boxes, 161
    - adding items to lists, 163
    - clearing lists, 165-166
    - manipulating items at design time, 162
    - manipulating items at run-time, 162-168
    - removing items from lists, 164-165
    - retrieving information from selected items in lists, 166-167
    - sorting lists, 167-168
  - List View control, 182-187
  - Menu Strip control, 196-198
  - nonvisual controls, 20-21
  - OpenFileDialog control, 409-415
    - creating file filters, 412
    - displaying Open File dialog, 412-413
  - OpenFileDialog controls, 17, 20, 23, 26
  - Panel control, 157-158
  - panels, 157-158
  - Picturebox controls, 17-19
  - SaveFileDialog control, 409, 413-415
  - selecting groups of controls, 123-125
  - sizing, 126-130
  - Status Bar control, 213-214
  - Tab control, 177-180

- tab orders, creating, 131-133
- tabbed dialog boxes, creating, 177-180
- text boxes, 146
  - adding scrollbars to, 150
  - aligning text in, 148
  - common events, 153
  - creating multiline text boxes, 148-149
  - creating password fields, 152
  - limiting number of characters entered, 151
- Timer control, 174-176
- Toolbar control, 211
- ToolStrip control, 208-210
- Tree View control, 187-190
- two button controls, 17
- visible controls, adding to forms, 18

**Controls collections, 73-75**

**ControlText system**

- color, 395

**Copy() method, 417**

**copying files, 416-417**

**CreateDirectory() method, 424**

**CreateGraphics**

- method, 68-69

**CreateSubKey() method, 431**

**custom dialog boxes, creating, 373-375**

**customizing Visual Basic, 32**

- docking design windows, 34-36
- floating design windows, 34
- hiding/displaying design windows, 33-36

**D****DashStyle property, 392****data types, 238**

- Boolean data type, 239-240
- Byte data type, 239
- casting data between data types, 241-242
- Char data type, 239
- Date data type, 239-240, 283-284
- Decimal data type, 239-240
- determining, 238
- Double data type, 239-240
- Integer data type, 239-240
- Long data type, 239-240
- naming conventions, 259-260
- Object data type, 239-240
- prefixes, 259-260
- Registry (Windows), 429
- SByte data type, 239
- selecting, 240
- Short data type, 239-240
- Single data type, 239-240
- String data type, 239-240
- UInteger data type, 239
- ULong data type, 239
- UShort data type, 239

**DataAdapter objects, 452, 456-457****databases, 451-452**

- ADO.NET, 452
  - closing data source connections, 455
- DataAdapter objects, 452, 456-457

- database connections, 453-455
- DataRow, 459-460
- DataSet objects, 452
- DataTable objects, 452, 456-458
- SqlConnection objects, 452
- SqlDataAdapter objects, 456-457

**records**

- creating, 463-464
- deleting, 464
- editing, 462
- navigating, 460-462
- running, 465

**DataSet objects, 452****DataTable objects, 452, 456**

- creating, 458
- creating records, 463-464
- DataRow, referencing fields in, 459-460
- deleting records, 464
- editing records, 462
- navigating records, 460-462
- populating, 458

**Date data type, 239-240****date/time**

- adding time to specific dates, 285-286
- current system date/time, retrieving, 288
- Date data type, 283-284
- DateAdd() function, 285-286
- DateDiff() function, 286-287
- DatePart() function, 287

**date/time**

- DateTime structure, 284, 288
- file date/time information, retrieving, 420
- formatting, 287-288
- intervals between
  - dates/times, determining, 286-287
- IsDate() function, 289
- parts of dates, retrieving, 287
- subtracting time from specific dates, 285-286
- values as dates, determining if, 289
- Debug.WriteLine()**
  - method, 336
- debugging code, 323-324**
  - break points, 329-331
  - build errors, 326-328
  - comments, 324-325
  - error handlers
    - creating via
      - Try...Catch...Finally structures, 336-339
    - handling exceptions, 339-344
  - Immediate window, 331-335
  - On Error statements, 337
  - Registry deployments (Windows), 437
  - runtime errors, 326-328
- Decimal data type, 239-240**
- declaration statements (events), 84**
- declaring**
  - arrays, 251
  - objects, 360-361
  - variables, 244-246
    - explicit variable declaration, 247-248
    - static scope, 258-259
- Delete() method, 73, 418-419, 424**
- DeleteSubKey() method, 431**
- deleting**
  - DataTable object
    - records, 464
  - directories (folders), 424
  - files, 418-419
  - menu items, 200
  - objects, 496
  - Registry keys (Windows), 431
- design windows**
  - displaying, 33
  - docking, 34-36
  - floating, 34
  - hiding, 33-36
- Desktop system color, 395**
- dialog boxes**
  - custom dialog boxes, creating, 373-375
  - tabbed dialog boxes, creating, 177-180
- dimensioning**
  - arrays, 251
  - objects, 360-361
  - variables, 244, 246
- directories (folders)**
  - BaseDirectory() method, 444
  - CreateDirectory() method, 424
  - creating, 424
  - deleting, 424
  - Directory object, 424
  - existence of, determining, 424
  - moving, 424
- displaying**
  - design windows, 33
  - forms
    - maximized state, 109-110
    - minimized state, 109-110
    - normal state, 109-110
    - specific initial position, 111
  - lists via list boxes, 161
    - adding items to lists, 163
    - clearing lists, 165-166
    - manipulating items at design time, 162
    - manipulating items at runtime, 162-168
    - removing items from lists, 164-165
    - retrieving information from selected items in lists, 166-167
    - sorting lists, 167-168
  - log files, 445-447
  - messages via
    - MessageBox.Show() function, 367-368
    - determining which button is clicked, 372
    - guidelines for creating good messages, 373
    - specifying buttons/icons, 369-371

- Open File dialog, 412-413
- properties (objects), 11
- Registry options (Windows), 434
- static text via Label control, 145-147
- text files, 445-447
- toolbars, 37
- windows, 10

**Dispose() method, 71**

**distributable components, defining, 6**

**distributed applications, uninstalling, 486-487**

**dithered colors, 44**

**division operations, 271**

**DLL, 9**

**Do...Loop loops**

- Continue Do statements, 317
- creating, 316-320
- ending, 316-317
- example of, 318-320

**docking**

- design windows, 34-36
- toolbars, 38

**Double data type, 239-240**

**double-clicking in Visual Studio 2010, 10**

**drag handles (toolbars), 38**

**DrawEllipse() method, 398**

**drawing controls, 119**

**DrawLine() method, 397**

**DrawRectangle() method, 70-71, 398**

**DrawString() method, 399**

**drop-down lists**

- creating via combo boxes, 168-170
- toolbar buttons, 212

**early binding, 358-360**

**editing DataTable object records, 462**

## **E**

**ellipses, drawing, 398**

**Else statements, 296**

**Elseif statements, 297-299**

**Enabled property, 149**

**encapsulating code/data via classes, 348**

**End If statements, 23-24, 274, 295**

- Else statements, 296
- False expressions, 296

**ending programs, writing code for, 24-25**

**enhanced lists, creating, 182**

- columns, 183
- list items
  - adding, 183-186
  - determining selected items, 186
  - removing, 186-187

**error handlers**

- creating via Try...Catch...Finally structures, 336-339
- exceptions, handling, 339-344

**events, 22**

- build example, 87
- event handler creation, 89-92
- user interface creation, 88

**Click events, 24-25, 81, 84, 153-155, 382, 477**

**declaration statements, 84**

**event handlers, creating, 22, 89-90, 92**

**event-driven programming, 79**

**FormClosed events, 455**

**invoking (triggering), 80**

- objects, 81

- OS (operating systems), 82

- user interaction, 81

**methods versus, 80**

**MouseDown events, 84-86, 153, 382**

**MouseEnter events, 382**

**MouseHover events, 382**

**MouseLeave events, 382**

**MouseMove events, 90, 153, 382**

**MouseUp events, 153, 382**

**MultilineChanged events, 81**

**names, 92**

**object events, accessing, 83-85**

**Paint events, 82, 405**

**parameters, 84-87**

**recursive events, 82**

**Resize events, 405**

**TextChanged event, 81, 153**

**Excel (MS) automation****Excel (MS) automation, 470**

- cell manipulation in workbooks, 473-475
  - forcing showing of Excel, 473
  - instances of automation server creation, 472
  - library reference creation, 470-472
  - testing applications, 475-476
  - workbook creation, 473
- executable components, 9**
  - existing projects, opening, 32**
  - Exists() method, 416, 424**
  - Exit For statements, 312**
  - Exit Try statements, 340**
  - exiting For...Next loops early, 312**
  - explicit variable declaration, 247-248**
  - exponentiation operations, 271**
  - expressions, variables in, 246-247**

**F**

- False expressions, 296**
- file filters, creating, 412**
- files**
  - attributes
    - attribute flags, 421
    - retrieving, 420-421
  - browsing, 22-24
  - copying, 416-417
  - date/time information,
    - retrieving, 420
  - deleting, 418-419
  - existence of, determining, 416
  - File object, 415-423

- log files
  - creating, 443-444
  - displaying, 445-447
  - testing, 447
- moving, 417-418
- naming, 418
- properties, retrieving, 422-423
- renaming, 418
- text files
  - creating, 443-444
  - displaying, 445-447
  - reading, 441-442
  - testing, 447
  - writing, 439-441
- Filter property, 21**
- filters, creating, 412**
- finding help, 53**
- floating design windows, 34**
- folders (directories)**
  - BaseDirectory() method, 444
  - CreateDirectory() method, 424
  - creating, 424
  - deleting, 424
  - Directory object, 424
  - existence of, determining, 424
  - moving, 424

**Font objects, 399****Font property, 42****For...Next loops, 309**

- closing, 310
- Continue For statements, 312
- creating, 312-315
- example of, 312-315
- Exit For statements, 312

- exiting early, 312
- For statements, 310
- initiating, 310
- Next statements, 310
- specifying increment values, 311
- Step statements, 311

**formatting date/time, 287-288****FormBorderStyle property, 105-107****FormClosed events, 455****forms, 58**

- backgrounds
  - adding images to, 100-101
  - changing color, 98-99
- borders, 105-107
- closing, 112-113
- control box buttons, adding, 103-104
- controls, adding, 16-21, 38-39
  - double-clicking controls in toolbox, 118
  - dragging controls from toolbox, 118
  - drawing controls, 119
- controls, organizing
  - aligning controls, 125-126
  - anchoring controls, 128-130
  - autosizing controls, 128-130
  - creating tab orders, 131-133
  - grid (size and snap), 120-121

**grid (size and snap)**

- layering controls, 133
- selecting groups of controls, 123-125
- setting property values for
  - groups of controls, 127
- sizing controls, 126
- Snap to Lines, 122-123
- spacing groups of controls, 126
- defining, 9, 48, 95
- displaying in
  - maximized state, 109-110
  - minimized state, 109-110
  - normal state, 109-110
  - specific initial position, 111
- Graphics objects, 390
- graphics, persisting
  - on forms, 400
- icons, assigning, 14, 103
- maximize buttons, adding, 103-104
- MDI (multiple-document interface) forms, 136-139
- menus, 196-198
  - adding buttons to toolbars, 208-210
  - assigning shortcut keys to menu items, 206-207
  - button drop-down menus, 212
  - checked menu items, 200-201
  - context menus, 204-206
  - creating menu items, 199
  - deleting menu items, 200
  - moving menu items, 200

- programming, 202-204
- programming
  - toolbars, 211
- status bars, 213-214
- toolbars, 208-212
- minimize buttons, adding, 103-104
- modality, 108
- naming, 96
- nonmodal windows, 134
- scrollable forms, 134-136
- showing, 107-108
- sizing, 15, 107-110
- startup forms,
  - configuring, 140
- taskbar, preventing forms
  - from appearing in, 112
- text bars, 97-98
- title bar, 13
- topmost nonmodal windows, 134
- transparent forms, 134
- unloading, 112-113
- Windows Forms Applications, 8-9

**FormulaR1C1 property, 474****functions. See also methods**

- methods, exposing functions as, 356
- OpenPicture() function, 443

**G****Get construct, 354****GetAttributes() method, 420-421****GetValue() method, 432****global (namespace) scope, 256-257****GoTo statements, 304-305****graphics**

- adding images to backgrounds, 100-101
- circles, drawing, 398
- Clear() method, 69
- CreateGraphics method, 68-69
- creating, 400-406
- Dispose() method, 71
- DrawRectangle() method, 70-71
- ellipses, drawing, 398
- example of, 400-406
- forms, persisting
  - graphics on, 400
- Graphics object, 389-391
- Image List control, storing images in, 180-181
- lines, drawing, 397
- pens, 392-393
- rectangles, drawing, 396-398
- system colors, 393-396
- text, drawing, 399

**GrayText system color, 395****grid (size and snap)**

- GridSize property, 120
- LayoutMode property, 120
- organizing controls on forms, 120-121
- ShowGrid property, 120-122
- SnapToGrid property, 120-122

## Group Box control

Group Box control, 157-158

group boxes, 157-158

grouping controls, 123-125

## H

handlers (event), creating, 89-92

Height property, 15

help

context-sensitive help, 53

finding, 53

further reading, 497

hiding

design windows, 33, 36

Excel (MS), 473

toolbars, 37

windows, 10

hierarchical lists,

creating, 187

adding nodes, 188-189

clearing all nodes, 190

removing nodes, 190

Highlight system color, 395

HighlightText system color, 395

hives (Registry), 428

HKEY\_CLASSES\_ROOT

node, 428

HKEY\_CURRENT\_CONFIG

node, 428

HKEY\_CURRENT\_USER node,

428, 431

HKEY\_LOCAL\_MACHINE node,

428, 431

HKEY\_USERS node, 428

hotkeys, 198

## I

icons

forms, assigning icons to, 14, 103

user messages, specifying icons in, 369-371

IDE (integrated development environment), 7

If statements, 274

If...End constructs, 435

If...Then constructs, 274, 293-294

Else statements, 296

Elseif statements, 297

End If statements, 295

expressions

Elseif statements, 299

evaluating expressions for multiple values, 298

executing False expressions, 296

False expressions, 296

IsNumeric() function, 294

nesting, 297-298

IL (Intermediate Language), 493-494

images. *See* graphics

Immediate window, 331-335

InactiveBorder system

color, 395

InactiveCaption system

color, 395

InactiveCaptionText system

color, 395

infinite recursion, procedures

and, 232

Inflate() method, 397

initializing variables, 262-265

InputBox() function, 377-379

installation (setup) programs

ClickOnce technology and, 481-484, 488

Publish Wizard (ClickOnce technology), 482, 484, 488

testing, 486

instantiating objects, 68

binding references to variables, 357-361

releasing object references, 362

instantiating objects

Instr() function, 281-282

Integer data type, 239-240

IntelliSense, 62

interaction (program/user)

custom dialog boxes, creating, 373-375

keyboards, 379-382

MessageBox.Show() function, displaying messages via, 367-368

determining which button is clicked, 372

guidelines for creating good messages, 373

specifying buttons/icons, 369-371

mouse events, 382-385

user information, obtaining, 377-379

Invalidate() method, 405

invisible-at-runtime controls, 20-21

invoking (triggering)

events, 80

objects, 81

**MaxLength property**

OS (operating systems), 82  
 user interaction, 81  
 methods (objects), 66-67  
**IsDate() function, 289**  
**IsNumeric() function, 294**  
 iterative processing, 72

**J-K-L**

keyboards, user/program  
 interaction, 379-382

**Label control, 145-147**

labels (code), 304

late binding, 358-359

layering controls, 133

**LayoutMode property, 120**

**Len() function, 279**

libraries

defining, 76  
 type (object) libraries, 470

**line continuation characters, 71**

**lines, drawing, 397**

**list boxes, 161. See also lists**

design time, manipulating  
 items at, 162  
 runtime, manipulating items  
 at, 162-168

**lists**

adding items to, 163  
 clearing, 165-166  
 drop-down lists, creating via  
 combo boxes, 168-170  
 enhanced lists, creating, 182  
 adding list items, 183-186  
 columns, 183

determining select list  
 items, 186

removing list items,  
 186-187

hierarchical lists,  
 creating, 187  
 adding nodes, 188-189  
 clearing all nodes, 190  
 removing nodes, 190

List View control, 182-187

removing items from,  
 164-165

retrieving information from  
 selected items, 166-167  
 sorting, 167-168

Tree View control, 187-190

**literal values, passing to vari-  
 ables, 246**

**local (procedure-level) scope, 255**

**log files**

creating, 443-444  
 displaying, 445-447  
 testing, 447

**Long data type, 239-240**

**loops**

Do...Loop loops  
 Continue Do  
 statements, 317  
 creating, 316-320  
 ending, 316-317  
 example of, 318-320

For...Next loops, 309

closing, 310  
 Continue For  
 statements, 312  
 creating, 312-315

example of, 312-315  
 Exit For statements, 312  
 exiting early, 312  
 For statements, 310  
 initiating, 310  
 Next statements, 310  
 specifying increment val-  
 ues, 311  
 Step statements, 311  
 iterative processing, 72  
 recursive loops, 232  
 While...End loops, 320

**M**

**magic numbers, 242**

**managing projects, 45-46**

**math operations**

addition, 270  
 comparison operators,  
 273-274  
 division, 271  
 exponentiation, 271  
 modulus arithmetic, 271  
 multiplication, 271  
 negation, 270  
 order of operator precedence,  
 determining,  
 272-273  
 subtraction, 270

**maximize buttons, adding to  
 forms, 103-104**

**maximized state, displaying  
 forms in, 109-110**

**MaximumSize property, 107**

**MaxLength property, 151**

**MDI (multiple-document interface) forms**

- MDI (multiple-document interface) forms, 136-139**
- Me.Close() statements, 25**
- memory leaks, 71**
- Menu system color, 395**
- menus**
  - context menus, 204-206
  - creating, 196-198
  - drop-down menus, 212
  - menu items
    - assigning shortcut keys to, 206-207
    - checked menu items, 200-201
    - creating, 199
    - deleting, 200
    - moving, 200
  - Menu Strip control, 196-198
  - programming, 202-204
  - status bars, 213-214
  - toolbars
    - adding buttons to toolbars, 208-210
    - button drop-down menus, 212
    - programming, 211
- MenuText system color, 395**
- MessageBox.Show() method, 52, 74, 302, 367-373**
- messages, displaying via MessageBox.Show() function, 367-368**
  - buttons
    - determining which button is clicked, 372
    - guidelines for creating good messages, 373
    - specifying, 369-371
  - icons, 369-371
- metadata, 496**
- methods**
  - Add(), 73
  - BaseDirectory(), 444
  - Clear(), 69, 398
  - Close(), 455
  - Copy(), 417
  - CreateDirectory(), 424
  - CreateGraphics(), 68-69
  - CreateSubKey(), 431
  - Debug.WriteLine(), 336
  - defining, 65
  - Delete(), 73, 418-419, 424
  - DeleteSubKey(), 431
  - Dispose(), 71
  - DrawEllipse(), 398
  - DrawLine(), 397
  - DrawRectangle(), 70-71, 398
  - DrawString(), 399
  - dynamism, 67
  - events versus, 80
  - Exists(), 416, 424
  - functions, exposing as methods, 356
  - GetAttributes(), 420-421
  - GetValue(), 432
  - Inflate(), 397
  - Invalidate(), 405
  - invoking (triggering), 66-67
  - MessageBox.Show(), 52, 74, 302, 367-373
  - Move(), 417-418, 424
  - ReadToEnd(), 442
  - SetValue(), 432
  - ShowDialog(), 23
  - SourceFileExists(), 416
  - Write(), 440
  - WriteLine(), 440-441
- Microsoft IL (Intermediate Language), 493-494**
- Microsoft.VisualBasic.Left() function, 279-280**
- Microsoft.VisualBasic.Right() function, 280**
- Mid() function, 280-281**
- minimize buttons, adding to forms, 103-104**
- minimized state, displaying forms in, 109-110**
- MinimumSize property, 107**
- modality of forms, 108**
- module-level scope, 255**
- modules**
  - class modules, 218
  - defining, 48, 217
  - standard modules, classes versus, 349
- modulus arithmetic operations, 271**
- monitors, Visual Studio 2010 resolution requirements, 10**
- mouse**
  - Click events, 382
  - MouseDown events, 382
  - MouseEnter events, 382
  - MouseHover events, 382
  - MouseLeave events, 382
  - MouseMove events, 382

MouseDown events, 382  
 user/program interaction,  
 382-385  
 Visual Studio 2010, double-  
 clicking in, 10

**MouseDown event, 84-86, 153**

**MouseMove event, 90, 153**

**MouseUp event, 153**

**Move() method, 417-418, 424**

#### **moving**

directories (folders), 424  
 files, 417-418  
 menu items, 200

**multidimensional arrays, 252-254**

**Multiline property, 148**

**multiline text boxes, creating,  
 148-149**

**MultilineChanged events, 81**

**multiplication operations, 271**

**My.Computer.Registry object,  
 430-433**

## **N**

**Name property, 11-13, 41**

**namespace (global) scope,  
 256-257**

**namespaces (.NET Framework),  
 452, 494-495**

#### **naming**

constants, 246  
 data types, 259-260  
 events, 92  
 files, 418  
 forms, 96  
 naming collisions, 494

objects, 11-13, 260

projects, 8

scope, 257, 260

variables, 246

#### **navigating**

DataTable object records,  
 460-462

Visual Basic, 32

Visual Studio 2010, 10

#### **negation operations, 270**

#### **nesting**

If...Then constructs,  
 297-298

Select Case constructs, 304

#### **.NET Framework, 491**

CLR (Common Language  
 Runtime), 492-493

common type systems, 496

IL (Intermediate Language),  
 493-494

namespaces, 494-495

objects, deleting, 496

Visual Basic's  
 relationship to, 6

**New Project dialog, 7-8, 30**

**Next statements, 310**

#### **nodes (list)**

adding, 188-189

clearing, 190

removing, 190

**nonmodal windows, 134**

**nonvisual controls, 20-21**

**normal state, displaying forms in,  
 109-110**

**Not operator, 276**

**numbers, magic, 242**

## **O**

**object (type) libraries, 470**

**Object data type, 239-240**

**object models, 469**

#### **objects**

ActiveCell objects, 474

building example, 67-69, 71

classifying, 11

CommandBuilder  
 objects, 457

control objects, 58

creating via classes, 350-356

DataAdapter objects, 452,  
 456-457

DataSet objects, 452

DataTable objects, 452,  
 456-458

creating records, 463-464

deleting records, 464

editing records, 462

navigating records,  
 460-462

populating, 458

referencing fields in  
 DataRows, 459-460

declaring, 360-361

defining, 11, 57-58

dimensioning, 360-361

Directory object, 424

#### **events**

accessing, 83-85

invoking (triggering), 81

File object, 415-423

Font objects, 399

form objects, 58

Graphics object, 389-391

**objects**

- instantiating, 68
    - binding references to variables, 357-361
    - releasing object references, 362
  - iterative processing, 72
  - libraries, 76
  - lifetime of, 362-363
  - methods. *See* methods
  - My.Computer.Registry object, 430-433
  - naming, 11-13, 260
  - .NET Framework, 496
  - Object Browser, 75-76
  - prefixes, 260
  - properties. *See* properties
  - Range objects, 474
  - scope, browsing, 76
  - selecting, 40
  - SqlConnection objects, 452-453
  - SqlDataAdapter objects, 456-457
  - Startup object, 140
  - StreamReader object, 441-442
  - StreamWriter object, 439-441
  - testing example, 72
  - Office (MS)**
    - Excel automation, 470-476
    - Word automation, 475-477
  - OLE controls. *See* user controls**
  - On Error statements, 337**
  - Opacity property, 134**
  - Open File dialog, displaying, 412-413**
  - OpenFileDialog control, 17, 20, 23, 26, 409-415**
    - file filters, creating, 412
    - Open File dialog, displaying, 412-413
  - opening existing projects, 32**
  - OpenPicture() function, 443**
  - operator precedence (arithmetic operations), 272-273**
  - Or operator, 276-277**
- P**
- Paint events, 82, 405**
  - Panel control, 157-158**
  - panels, 157-158**
  - parameters (events), 84-87**
    - code procedures, passing parameters in, 228-230
    - defining, 85
    - multiple parameters in events, 85
  - password fields, creating, 152**
  - PasswordChar property, 152**
  - pens, 392-393**
  - peripherals**
    - keyboards, 379-382
    - mouse, 382-385
  - PictureBox controls, 17-19**
  - pictures. *See* graphics**
  - prefixes**
    - data types, 259-260
    - objects, 260
    - scope, 260
  - procedure-level (local) scope, 255**
  - procedures, 51**
    - calling, 225-230
    - creating, 219-220
      - declaring procedures with return values, 224-225
      - declaring procedures without return values, 220-224
    - exiting, 231-232
    - infinite recursion, 232
  - program interaction**
    - custom dialog boxes, creating, 373-375
    - keyboards, 379-382
    - MessageBox.Show() function, displaying messages via, 367-368
      - determining which button is clicked, 372
      - guidelines for creating good messages, 373
      - specifying buttons/icons, 369-371
    - mouse events, 382-385
    - user information, obtaining, 377-379
  - programs**
    - defining, 47
    - projects versus, 47
    - running, 25-27
    - terminating, 24-25
  - projects**
    - components of, 47-48
    - creating, 7-9, 30-31
    - defining, 6, 47
    - DLL, 9
    - executable components, 9

- files
  - adding, 49-50
  - removing, 49-50
- grouping, 47
- managing, 45-46
- naming, 8
- New Project dialog, 7-8, 30
- opening existing projects, 32
- programs versus, 47
- properties, 48-49
- Recent Projects dialog, 30
- saving, 13-14

**properties (objects), 40**

- AcceptButton, 155-156
- AcceptsReturn, 150
- Auto Hide, 36
- BackColor, 41, 99-100
- BackgroundImage, 100-102
- BorderStyle, 41
- button creation, 61-64
- CancelButton, 156
- changing, 40-42
- CheckState, 156
- classes, adding to, 352-353
- color properties, 42-44
- ConnectionString, 454-455
- defining, 11, 58
- descriptions, viewing, 44
- displaying, 11
- Enabled, 149
- Filter, 21
- Font, 42
- FormBorderStyle, 105-107
- FormulaR1C1, 474
- GridSize, 120

- Height, 15
- LayoutMode, 120
- MaximumSize, 107
- MaxLength, 151
- MinimumSize, 107
- Multiline, 148
- Name, 11-13, 41
- Opacity, 134
- PasswordChar, 152
- read-only properties, 60, 355-356
- readable properties, 354
- referencing, 59-60
- ScrollBars, 150
- setting, 48-49
- ShowGrid, 120-122
- ShowInTaskbar, 112
- Size, 42
- SnapToGrid, 120-122
- StartPosition, 109-111
- TabIndex, 132-133
- TabStop, 133
- Text, 13, 24
- TextAlign, 148
- unchangeable properties, 60
- viewing, 40
- Width, 15
- WindowState, 109-110
- WordWrap, 150
- writable properties, creating, 354-355
- write-only properties, creating, 355-356

**Properties window (Visual Studio 2010), 10**

**Publish Wizard, 482-484, 488**

## Q-R

**Quit button, 24-25**

**radio buttons, 159-160**

**Range objects, 474**

**read-only properties, 60, 355-356**

**readable properties, 354**

**reading text files, 441-442**

**ReadToEnd() method, 442**

**Recent Projects dialog, 30**

**rectangles, drawing, 396-398**

**recursive events, 82**

**recursive loops, 232**

### referencing

- array variables, 251-252

- automation libraries

- Excel (MS), 470-472

- Word (MS), 476-477

- constants, 243

### Registry (Windows), 427

- accessing via

- My.Computer.Registry

- object, 430

- creating Registry keys, 430-431

- deleting Registry keys, 431

- getting Registry key values, 432-433

- setting Registry key values, 432-433

- data types, 429

- deployments, 437

- displaying options from, 434

- hives, 428

- HKEY\_CLASSES\_ROOT

- node, 428

## Registry (Windows)

- HKEY\_CURRENT\_CONFIG
    - node, 428
  - HKEY\_CURRENT\_USER node, 428, 431
  - HKEY\_LOCAL\_MACHINE node, 428, 431
  - HKEY\_USERS node, 428
  - keys
    - creating, 430-431
    - deleting, 431
    - getting values, 432-433
    - setting values, 432-433
    - saving options to, 435
    - stored options, 435-436
    - structure of, 428-429
  - removing**
    - items from lists (list boxes), 164-165
    - list items from enhanced lists, 186-187
    - nodes from hierarchical lists, 190
    - project files, 49-50
  - renaming files, 418**
  - Replace() function, 283**
  - reserved words, 246**
  - Resize events, 405**
  - resolution (monitors), 10**
  - running programs, 25-27**
  - runtime errors, 326-328**
- S**
- SaveFileDialog control, 409, 413-415
  - saving
    - projects, 13-14
    - Registry options (Windows), 435
  - SByte data type, 239**
  - scope**
    - block (structure) scope, 254-255
    - browsing, 76
    - defining, 254
    - global (namespace) scope, 256-257
    - limiting, 258
    - module-level scope, 255
    - name conflicts, 257
    - naming conventions, 260
    - prefixes, 260
    - procedure-level (local) scope, 255
    - static scope, 258-259
  - scrollable forms, 134-136**
  - scrollbars, 150**
  - Select Case constructs**
    - Case Else statements, 302
    - Case statements, 299-302
    - example of, 300-303
    - MessageBox.Show() statements, 302
    - nesting, 304
    - uses for, 303-304
    - values, evaluating more than one, 299-300
  - selecting groups of controls, 123-125**
  - servers, defining, 469
  - Set construct, 354-355**
  - setup (installation) programs**
    - ClickOnce technology and, 481-484, 488
    - testing, 486
  - SetValue() method, 432**
  - shapes, drawing**
    - circles, 398
    - ellipses, 398
    - lines, 397
    - rectangles, 396-398
  - Short data type, 239-240**
  - shortcut keys, assigning to menu items, 206-207**
  - ShowDialog() method, 23**
  - ShowGrid property, 120-122**
  - showing forms, 107-108**
  - ShowInTaskbar property, 112**
  - Single data type, 239-240**
  - size and snap (grid)**
    - GridSize property, 120
    - LayoutMode property, 120
    - organizing controls on forms, 120-121
    - ShowGrid property, 120-122
    - SnapToGrid property, 120-122
  - Size property, 42**
  - sizing**
    - controls, 126-130
    - docked design windows, 35
    - forms, 15, 107-110
    - MaximumSize property, 107
    - MinimumSize property, 107
    - toolbars, 38
  - Snap to Lines property, 122-123**
  - SnapToGrid property, 120-122**
  - Solution Explorer, 45-46**

**solutions**

- creating, 47
- defining, 6, 47

**sorting lists (list boxes), 167-168****SourceFileExists() method, 416****spaces, removing from beginning and end of strings, 282****spacing groups of controls, 126****SqlConnection objects, 452-453****SqlDataAdapter objects, 456-457****standard modules, classes versus, 349****Start page (Visual Studio 2010), 7-8, 30****starting Visual Studio 2010, 7****StartPosition property, 109-111****startup forms, configuring, 140****Startup object, 140****statements. See specific statements****static scope, 258-259****static text, displaying via Label control, 145-147****Status Bar control, 213-214****Step statements, 311****stopping programs, 24-25****stored Registry options (Windows), 435-436****storing images in Image List control, 180-181****StreamReader object, 441-442****StreamWriter object, 439-441****strict typing, 247-250****String data type, 239-240****strings****concatenating, 278-279**

- number of characters in, determining, 279
- replacing text within, 283

**retrieving text from**

- left side, 279-280
- right side, 280
- within strings, 280-281

**spaces, removing from beginning and end of strings, 282****strings within strings, determining, 281-282****structure (block) scope, 254-255****subtraction operations, 270****support**

- context-sensitive help, 53
- finding, 53
- further reading, 497

**system colors, 393-396****system date/time, retrieving, 288****System namespace, 494****System.Data namespace, 452****T****Tab control, 177, 179-180****tab order (controls), 131-133****tabbed dialog boxes, creating, 177-180****TabIndex property, 132-133****TabStop property, 133****taskbar**

- forms, preventing from appearing in taskbar, 112
- ShowInTaskbar property, 112

**technical support**

- context-sensitive help, 53
- finding, 53
- further reading, 497

**terminating programs, 24-25****testing**

- Excel automation, 475-476
- installation (setup) programs, 486
- log files, 447
- Registry deployments (Windows), 437
- text files, 447

**text****ActiveCaptionText system color, 394****ControlText system color, 395****drawing via DrawString() method, 399****Font objects, 399****Font property, 42****GrayText system color, 395****HighlightText system color, 395****InactiveCaptionText system color, 395****MaxLength property, 151****MenuText system color, 395****static text, displaying via Label control, 145-147****strings**

- replacing text within strings, 283

- retrieving text from left side of strings, 279-280

- retrieving text from right side of strings, 280

**text**

- retrieving text from within strings, 280-281
  - text bars (forms), displaying text on, 97-98
  - text files
    - creating, 443-444
    - displaying, 445-447
    - reading, 441-442
    - testing, 447
    - writing, 439-441
  - TextAlign property, 148
  - text bars, displaying text on, 97-98**
  - text boxes, 146**
    - aligning text in, 148
    - common events, 153
    - limiting number of characters entered, 151
    - multiline text boxes, creating, 148-149
    - password fields, creating, 152
    - scrollbars, adding to, 150
  - Text property, 13, 24**
  - Textbox controls, 81**
  - TextChanged event, 81, 153**
  - time/date**
    - adding time to specific dates, 285-286
    - current system date/time, retrieving, 288
    - Date data type, 283-284
    - DateAdd() function, 285-286
    - DateDiff() function, 286-287
    - DatePart() function, 287
    - DateTime structure, 284, 288
    - file time/date information, retrieving, 420
    - formatting, 287-288
    - intervals between times/dates, determining, 286-287
    - IsDate() function, 289
    - parts of dates, retrieving, 287
    - subtracting time from specific dates, 285-286
    - values as dates, determining if, 289
  - Timer control, 81, 174-176**
  - title bar (forms), 13**
  - toolbars**
    - displaying, 37
    - docking, 38
    - drag handles, 38
    - hiding, 37
    - sizing, 38
    - Toolbar control, 211
    - ToolStrip control, 208-210
  - toolbox**
    - closing, 10
    - double-clicking controls in toolbox, 118
    - dragging controls from toolbox, 118
    - opening, 10
  - transparent forms, 134**
  - triggering (invoking)**
    - events, 80
    - objects, 81
    - OS (operating systems), 82
    - user interaction, 81
    - methods (objects), 66-67
  - True statements, 274**
  - Try...Catch...Finally structures, creating error handlers, 336-339**
  - Try...End Try structures, 337-340**
  - two button controls, 17**
  - type (object) libraries, 470**
- ## U
- UInteger data type, 239**
  - ULong data type, 239**
  - uninstalling applications, 486-487**
  - unloading forms, 112-113**
  - user controls, 48**
  - user interaction**
    - custom dialog boxes, creating, 373-375
    - keyboards, 379-382
    - MessageBox.Show() function, displaying messages via, 367-368
    - determining which button is clicked, 372
    - guidelines for creating good messages, 373
    - specifying buttons/icons, 369-371
    - mouse events, 382-385
    - user information, obtaining, 377-379
  - user interfaces, creating (event building example), 88**
  - UShort data type, 239**

**V****variables, 60**

- creating, 261-262
- declaring, 244-246
  - explicit variable declaration, 247-248
  - static scope, 258-259
- defining, 237
- dimensioning, 244-246
- expressions, 246-247
- initializing, 262-265
- naming, 246
- object variables, binding, 357
  - creating new objects, 360-361
  - early binding, 360
  - late binding, 358-359
- passing literal values to, 246
- referencing, 251-252
- storing values in, 51
- strict typing, 247-250

**viewing properties (objects), 40, 44****visible controls, adding to forms, 18****Visual Basic**

- customizing, 32
  - docking design windows, 34-36

- floating design windows, 34
  - hiding/displaying design windows, 33-36
- navigating, 32
- .NET Framework, Visual Basic's relationship to, 6

**Visual Studio 2010, 7**

- closing windows, 10
- displaying windows, 10
- double-clicking in, 10
- hiding windows, 10
- monitor resolution requirements, 10
- navigating, 10
- Properties window, 10
- Start page, 7-8, 30
- starting, 7
- Toolbox window, 10

**W-X-Y-Z****While...End loops, 320****Width property, 15****Window system color, 395****windows, 10****Windows Forms****Applications, 8-9****Windows Registry. See Registry****WindowState property, 109-110****wizards, Publish Wizard, 482-484, 488****Word (MS) automation, 475-477****WordWrap property, 150****workbooks (Excel)**

- cell manipulation in, 473-475
- creating, 473

**writable properties (objects), 354-355****Write() method, 440****write-only properties (objects), 355-356****WriteLine() method, 440-441****writing**

- code, 21
  - browsing files example, 22-24
  - terminating programs example, 24-25
- text files, 439-441

**Xor operator, 276-277****yes/no options via check boxes, 156-157**