

James Foxall

Sams **Teach Yourself**

# Visual Basic® 2012

in **24**  
**Hours**

**SAMS**

FREE SAMPLE CHAPTER



SHARE WITH OTHERS

James Foxall

Sams **Teach Yourself**

# **Visual Basic 2012**

in **24**  
**Hours**

**SAMS**

800 East 96th Street, Indianapolis, Indiana, 46240 USA

## **Sams Teach Yourself Visual Basic 2012 in 24 Hours, Complete Starter Kit**

Copyright © 2013 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: 978-0-672-33629-4

ISBN-10: 0-672-33629-4

Library of Congress Cataloging-in-Publication Data is on file.

Printed in the United States of America

First Printing October 2012

### **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**

Greg Wiegand

**Executive Editor**

Neil Rowe

**Development Editor**

Mark Renfrow

**Managing Editor**

Sandra Schroeder

**Project Editor**

Mandie Frank

**Copy Editor**

Margo Catts

**Indexer**

Cheryl Lenser

**Proofreader**

Sarah Kearns

**Technical Editor**

J. Boyd Nolan

**Publishing**

**Coordinator**

Cindy Teeters

**Multimedia**

**Developer**

Tim Warner

**Designer**

Gary Adair

**Composition**

TnT Design, Inc.

# Contents at a Glance

Introduction .....	1
<b>PART I</b> The Visual Basic 2012 Environment	
<b>HOUR 1</b> Jumping in with Both Feet: A Visual Basic 2012 Programming Tour .....	5
<b>2</b> Navigating Visual Basic 2012 .....	29
<b>3</b> Understanding Objects and Collections .....	57
<b>4</b> Understanding Events .....	79
<b>PART II</b> Building a User Interface	
<b>5</b> Building Forms: The Basics .....	95
<b>6</b> Building Forms: Advanced Techniques .....	117
<b>7</b> Working with Traditional Controls .....	145
<b>8</b> Using Advanced Controls .....	173
<b>9</b> Adding Menus and Toolbars to Forms .....	195
<b>PART III</b> Making Things Happen—Programming	
<b>10</b> Creating and Calling Code Procedures .....	217
<b>11</b> Using Constants, Data Types, Variables, and Arrays .....	237
<b>12</b> Performing Arithmetic, String Manipulation, and Date/Time Adjustments .....	269
<b>13</b> Making Decisions in Visual Basic Code .....	293
<b>14</b> Looping for Efficiency .....	309
<b>15</b> Debugging Your Code .....	323
<b>16</b> Designing Objects Using Classes .....	347
<b>17</b> Interacting with Users .....	367
<b>18</b> Working with Graphics .....	389
<b>PART IV</b> Working with Data	
<b>19</b> Performing File Operations .....	409
<b>20</b> Working with the Registry and Text Files .....	427
<b>21</b> Working with a Database .....	451
<b>22</b> Controlling Other Applications Using Automation .....	469
<b>PART V</b> Deploying Solutions and Beyond	
<b>23</b> Deploying Applications .....	481
<b>24</b> The 10,000-Foot View .....	491
Index .....	499

# Table of Contents

Introduction	1
<b>PART I: The Visual Basic 2012 Environment</b>	
<b>HOURL 1: Jumping in with Both Feet: A Visual Basic 2012 Programming Tour</b>	<b>5</b>
Starting Visual Basic 2012	6
Creating a New Project	7
Understanding the Visual Studio 2012 Environment	10
Changing the Characteristics of Objects	11
Adding Controls to a Form	16
Designing an Interface	17
Writing the Code Behind an Interface	21
Running a Project	25
<b>HOURL 2: Navigating Visual Basic 2012</b>	<b>29</b>
Using the Visual Basic 2012 Start Page	30
Navigating and Customizing the Visual Basic Environment	32
Working with Toolbars	37
Adding Controls to a Form Using the Toolbox	38
Setting Object Properties Using the Properties Window	39
Managing Projects	45
A Quick-and-Dirty Programming Primer	51
Getting Help	53
<b>HOURL 3: Understanding Objects and Collections</b>	<b>57</b>
Understanding Objects	58
Understanding Properties	58
Understanding Methods	65
Building a Simple Object Example Project	67
Understanding Collections	72
Using the Object Browser	75

<b>HOURL 4: Understanding Events</b>	<b>79</b>
Understanding Event-Driven Programming .....	79
Building an Event Example Project .....	87
Keeping Event Names Current .....	92
<b>PART II: Building a User Interface</b>	
<b>HOURL 5: Building Forms: The Basics</b>	<b>95</b>
Changing a Form's Name .....	96
Changing a Form's Appearance .....	97
Showing and Hiding Forms .....	107
<b>HOURL 6: Building Forms: Advanced Techniques</b>	<b>117</b>
Working with Controls .....	117
Creating Topmost Nonmodal Windows .....	134
Creating Transparent Forms .....	135
Creating Scrollable Forms .....	135
Creating MDI Forms .....	137
Setting the Startup Form .....	140
<b>HOURL 7: Working with Traditional Controls</b>	<b>145</b>
Displaying Static Text with the Label Control .....	145
Allowing Users to Enter Text Using a Text Box .....	146
Creating Buttons .....	153
Creating Containers and Groups of Option Buttons .....	157
Displaying a List with the List Box .....	161
Creating Drop-Down Lists Using the Combo Box .....	168
<b>HOURL 8: Using Advanced Controls</b>	<b>173</b>
Creating Timers .....	174
Creating Tabbed Dialog Boxes .....	177
Storing Pictures in an Image List Control .....	180
Building Enhanced Lists Using the List View Control .....	182
Creating Hierarchical Lists Using the Tree View Control .....	187

## Sams Teach Yourself Visual Basic 2012 in 24 Hours

<b>HOUR 9: Adding Menus and Toolbars to Forms</b>	<b>195</b>
Building Menus .....	196
Using the Toolbar Control .....	207
Creating a Status Bar .....	213
<b>PART III: Making Things Happen—Programming</b>	
<b>HOUR 10: Creating and Calling Code Procedures</b>	<b>217</b>
Creating Visual Basic Code Modules .....	217
Writing Code Procedures .....	219
Calling Code Procedures .....	225
Exiting Procedures .....	231
Avoiding Infinite Recursion .....	232
<b>HOUR 11: Using Constants, Data Types, Variables, and Arrays</b>	<b>237</b>
Understanding Data Types .....	238
Defining and Using Constants .....	242
Declaring and Referencing Variables .....	244
Working with Arrays .....	250
Determining Scope .....	254
Declaring Variables of Static Scope .....	258
Naming Conventions .....	259
Using Variables in Your Picture Viewer Project .....	261
<b>HOUR 12: Performing Arithmetic, String Manipulation, and Date/Time Adjustments</b>	<b>269</b>
Performing Basic Arithmetic Operations with Visual Basic .....	270
Comparing Equalities .....	274
Understanding Boolean Logic .....	274
Manipulating Strings .....	278
Working with Dates and Times .....	283
<b>HOUR 13: Making Decisions in Visual Basic Code</b>	<b>293</b>
Making Decisions Using If . . . Then .....	293
Branching Within a Procedure Using GoTo .....	304

<b>HOURL 14: Looping for Efficiency</b>	<b>309</b>
Looping a Specific Number of Times Using For . . . Next	309
Using Do . . . Loop to Loop an Indeterminate Number of Times	315
<b>HOURL 15: Debugging Your Code</b>	<b>323</b>
Adding Comments to Your Code	324
Identifying the Two Basic Types of Errors	326
Using Visual Basic's Debugging Tools	328
Writing an Error Handler Using Try . . . Catch . . . Finally	336
<b>HOURL 16: Designing Objects Using Classes</b>	<b>347</b>
Understanding Classes	348
Instantiating Objects from Classes	357
<b>HOURL 17: Interacting with Users</b>	<b>367</b>
Displaying Messages Using the MessageBox.Show() Function	367
Creating Custom Dialog Boxes	373
Using InputBox() to Get Information from a User	377
Interacting with the Keyboard	379
Using the Common Mouse Events	382
<b>HOURL 18: Working with Graphics</b>	<b>389</b>
Understanding the Graphics Object	389
Working with Pens	392
Using System Colors	393
Working with Rectangles	396
Drawing Shapes	397
Drawing Text	399
Persisting Graphics on a Form	400
Building a Graphics Project Example	400
<b>PART IV: Working with Data</b>	
<b>HOURL 19: Performing File Operations</b>	<b>409</b>
Using the OpenFileDialog and SaveFileDialog Controls	409
Manipulating Files with the File Object	415
Manipulating Directories with the Directory Object	424



**Sams Teach Yourself Visual Basic 2012 in 24 Hours**

<b>HOOR 20: Working with the Registry and Text Files</b>	<b>427</b>
Working with the Registry .....	427
Reading and Writing Text Files .....	439
<b>HOOR 21: Working with a Database</b>	<b>451</b>
Introducing ADO.NET .....	452
Manipulating Data .....	456
<b>HOOR 22: Controlling Other Applications Using Automation</b>	<b>469</b>
Automating Microsoft Excel .....	470
Automating Microsoft Word .....	476
<b>PART V: Deploying Solutions and Beyond</b>	
<b>HOOR 23: Deploying Applications</b>	<b>481</b>
Understanding ClickOnce Technology .....	481
Using the Publish Wizard to Create a ClickOnce Application .....	482
Testing Your Picture Viewer ClickOnce Install Program .....	486
Uninstalling an Application You've Distributed .....	486
Setting Advanced Options for Creating ClickOnce Programs .....	488
<b>HOOR 24: The 10,000-Foot View</b>	<b>491</b>
The .NET Framework .....	491
Common Language Runtime .....	492
Microsoft Intermediate Language .....	493
Namespaces .....	494
Common Type System .....	496
Garbage Collection .....	496
Further Reading .....	497
<b>Index</b>	<b>499</b>

## About the Author

**James Foxall** is president of Tigerpaw Software, Inc. ([www.tigerpawsoftware.com](http://www.tigerpawsoftware.com)), a Bellevue, Nebraska, Microsoft Certified Partner specializing in software solutions for technology providers. Tigerpaw's award-winning business automation solution is designed to automate contact management, marketing, service and repair, proposal generation, inventory control, and purchasing. At the start of 2012, the current release of Tigerpaw had more than 35,000 licensed users. Foxall's experience in creating certified Office-compatible software has made him an authority on application interface and behavior standards of applications for the Microsoft Windows and Microsoft Office environments.

Foxall has been writing commercial production Visual Basic code for more than 14 years. He's the author of numerous books, including *Practical Standards for Microsoft Visual Basic* and *MCSD in a Nutshell: The Visual Basic Exams*. He also has written articles for *Access-Office-VBA Advisor* and *Visual Basic Programmer's Journal*. Foxall has a bachelor's degree in management of information systems (MIS) and a master's degree in Business Administration (MBA). He is a Microsoft Certified Solution Developer and an international speaker on programming technologies as well as business process improvements. James enjoys spending time with his family, playing guitar, listening to amazing bands, and playing computer games. You can reach him at [www.jamesfoxall.com](http://www.jamesfoxall.com).

## Dedication

*This book is dedicated to all of my great co-workers at Tigerpaw;  
thank you for your dedication and your passion!*

## Acknowledgments

I would like to thank all the great people at Sams for their input and hard work; this book would not be possible without them!

## **We Want to Hear from You!**

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

You can email or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books stronger.

*Please note that I cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail I receive, I might not be able to reply to every message.*

When you write, please be sure to include this book's title and author as well as your name and phone or email address. I will carefully review your comments and share them with the author and editors who worked on the book.

Email: consumer@sampublishing.com

Mail: Neil Rowe  
Executive Editor  
Sams Publishing  
800 East 96th Street  
Indianapolis, IN 46240 USA

## **Reader Services**

Visit our website and register this book at [www.sampublishing.com/register](http://www.sampublishing.com/register) for convenient access to any updates, downloads, or errata that might be available for this book.

# Introduction

Visual Basic 2012 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 2012 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 2012 programming—and there's a lot to learn. You'll

discover 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!

*This page intentionally left blank*

## HOUR 3

# Understanding Objects and Collections

---

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

- ▶ Understanding objects
- ▶ Getting and setting properties
- ▶ Triggering methods
- ▶ Understanding method dynamism
- ▶ Writing object-based code
- ▶ Understanding collections
- ▶ Using the Object Browser

In Hour 1, “Jumping in with Both Feet: A Visual Basic 2012 Programming Tour,” you were introduced to programming in Visual Basic by building a Picture Viewer project. You then spent Hour 2, “Navigating Visual Basic 2012,” digging into the integrated development environment (IDE) and learning skills critical to your success with Visual Basic. In this hour, you begin learning about an important programming concept: *objects*.

The term *object* as it relates to programming might have been new to you prior to this book. The more you work with Visual Basic, the more you'll hear about objects. Visual Basic 2012, unlike its early predecessors, is a true object-oriented language. This hour doesn't discuss object-oriented programming in any detail; object-oriented programming is a complex subject and well beyond the scope of this book. Instead, you'll learn about objects in a more general sense.

Everything you use in Visual Basic is an object, so understanding this material is critical to your success with Visual Basic. For example, forms are objects, as are the controls you place on a form. Pretty much every element of a Visual Basic project is an object and belongs to a collection of objects. All objects have attributes (called



*properties*), most have methods, and many have events. Whether creating simple applications or building large-scale enterprise solutions, you must understand what an object is and how it works. In this hour, you'll learn what makes an object an object, and you'll also learn about collections.

### **By the Way**

If you've listened to the programming press at all, you've probably heard the term *object-oriented*, and perhaps words such as *polymorphism*, *encapsulation*, and *inheritance*. In truth, these object-oriented features of Visual Basic are exciting, but they're far beyond this hour (or the last hour, for that matter). You'll learn a little about object-oriented programming in this book, but if you're really interested in taking your programming skills to the next level, you should buy a book dedicated to the subject after you've completed this book.

## **Understanding Objects**

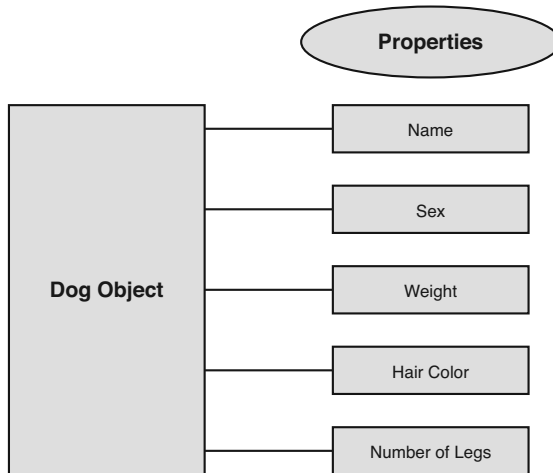
Object-oriented programming has been a technical buzzword for quite some time, but as far as Visual Basic programmers are concerned, it became a reality only with Visual Basic .NET. (No previous version of Visual Basic was a true object-oriented language.) Almost everywhere you look—the Web, publications, books—you read about objects. What exactly is an object? Strictly speaking, an *object* is a programming structure that encapsulates data and functionality as a single unit and for which the only public access is through the programming structure's interfaces (properties, methods, and events). In reality, the answer to this question can be somewhat ambiguous, because there are so many types of objects—and the number grows almost daily. All objects share specific characteristics, however, such as properties and methods.

The most commonly used objects in Visual Basic are the form object and the control object. Earlier hours introduced you to working with forms and controls and even showed you how to set form and control properties. In your Picture Viewer project from Hour 1, for example, you added a picture box and two buttons to a form. Both the PictureBox and the Button controls are *control objects*, but each is a specific type of control object. Another, less-technical example uses pets. Dogs and cats are definitely different entities (objects), but they both fit into the category of pet objects. Similarly, a text box and a button is each a unique type of object, but they're both considered control objects. This small distinction is important.

## **Understanding Properties**

All objects have attributes that are used to specify and return the state of the object. These attributes are properties, and you've already used some of them in previous hours in the Properties window. Indeed, every object exposes a specific set of properties,

but not every object exposes the same set of properties. To illustrate this point, I'll continue with the hypothetical pet object. Suppose that you have an object, and the object is a dog. This Dog object has certain properties common to all dogs. These properties include attributes such as the dog's name, the color of its hair, and even the number of legs it has. All dogs have these same properties; however, different dogs have different values for these properties. Figure 3.1 illustrates such a Dog object and its properties.



**FIGURE 3.1**  
Properties are the attributes that describe an object.

## Getting and Setting Properties

You've already seen how to read and change properties using the Properties window. The Properties window is available only at design time, however, and is used only to manipulate the properties of forms and controls. Most getting and setting of properties you'll perform will be done with Visual Basic code, not in the Properties window. When referencing properties in code, you specify the object's name first, followed by a period (.), and then the property name, as in the following syntax:

```
ObjectName.Property
```

If you had a Button object named `btnClickMe`, for example, you would reference the button's `Text` property this way:

```
btnClickMe.Text
```

This line of code would return whatever value was contained in the `Text` property of the Button object `btnClickMe`. To set a property to some value, you use an equals sign (=). To change the Button object's `Left` property, for example, you'd use a line of code such as the following:

```
btnClickMe.Left = 90
```

When you reference a property on the left side of an equals sign, you're setting the value. When you reference a property on the right side of the equals sign, you're getting (reading) the value. In the early days of BASIC, you actually used the word `Let` when setting values. This made the code easier to read for novices, but it was unnecessarily tedious. Nevertheless, using `Let` makes the statement clearer for this example, so I'll show the same code statement as before with the word `Let`:

```
Let btnClickMe.Left = 90
```

It's easier to see here that referencing the property on the left side of the equals sign indicates that you're setting the property to some value. The keyword `Let` is no longer a valid way to make variable assignments. If you enter a code statement that uses `Let`, you won't receive an error, but the code editor (also known as *the gremlins*) automatically removes the word `Let` from the statement for you.

The following line of code places the value of the `Left` property of the `Button` object called `btnClickMe` into a temporary variable. This statement retrieves the value of the `Left` property because the `Left` property is referenced on the right side of the equals sign:

```
intLeftVariable = btnClickMe.Left
```

Variables are discussed in detail in Hour 11, "Using Constants, Data Types, Variables, and Arrays." For now, think of a variable as a storage location. When the processor executes this statement, it retrieves the value in the `Left` property of the `Button` object `btnClickMe` and places it in the variable (storage location) titled `intLeftVariable`. Assuming that `btnClickMe`'s `Left` property value is `90`, as set in the previous example, the computer would process the code statement like this:

```
intLeftVariable = 90
```

Just as in real life, some properties can be read but not changed. Think back to the hypothetical pet object, and suppose that you have a `Gender` property to designate the gender of a `Dog` object. It's impossible for you to change a dog from a male to a female or vice versa (at least, I think it is). Because the `Gender` property can be retrieved but not changed, it's known as a *read-only* property. You'll often encounter properties that can be set in Design view but become read-only when the program is running.

One example of a read-only property is the `Height` property of the `Combo Box` control. Although you can view the value of the `Height` property in the Properties window, you can't change the value—no matter how hard you try. If you attempt to change the `Height` property using Visual Basic code, Visual Basic simply changes the value back to the default—eerie gremlins.

The best way to determine which properties of an object are read-only is to consult the online help for the object in question.

## Working with an Object and Its Properties

Now that you know what properties are and how they can be viewed and changed, you'll experiment with properties by modifying the Picture Viewer project you built in Hour 1. Recall from Hour 1 how you learned to set the Height and Width properties of a form in the Properties window. Here, you'll change the same properties, now using Visual Basic code.

You'll add two buttons to your Picture Viewer. One button will enlarge the form when clicked, and the other will shrink the form. This is a simple example, but it illustrates well how to change object properties in Visual Basic code.

Start by opening your Picture Viewer project from Hour 1. If you download the code samples from my site, I provide a Picture Viewer project for you to start with. Double-click `ViewerForm.vb` in the Solution Explorer window to show the form designer.

When the project first runs, the form has the Height and Width you specified in the Properties window. You'll add buttons to the form that a user can click to enlarge or shrink the form at runtime by following these steps:

1. Add a new button to the form by double-clicking the Button tool in the toolbox. Set the new button's properties as follows:

Property	Value
Name	btnEnlarge
Location	338,261
Size	21,23
Text	^ (Note: This is Shift+6.)

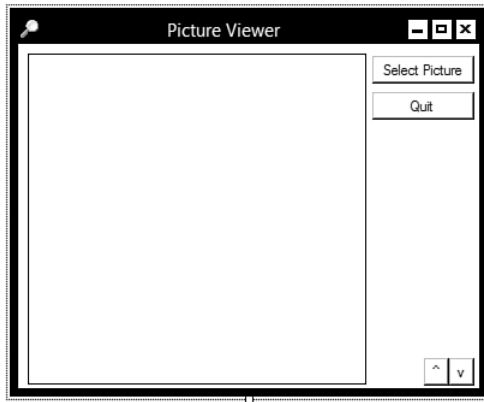
2. Now for the Shrink button. Again, double-click the Button tool in the toolbox to create a new button on the form. Set this new button's properties as follows:

Property	Value
Name	btnShrink
Location	359,261
Size	21,23
Text	v

Your form should now look like the one in shown in Figure 3.2.

**FIGURE 3.2**

Each button is an object, as is the form on which the buttons sit.

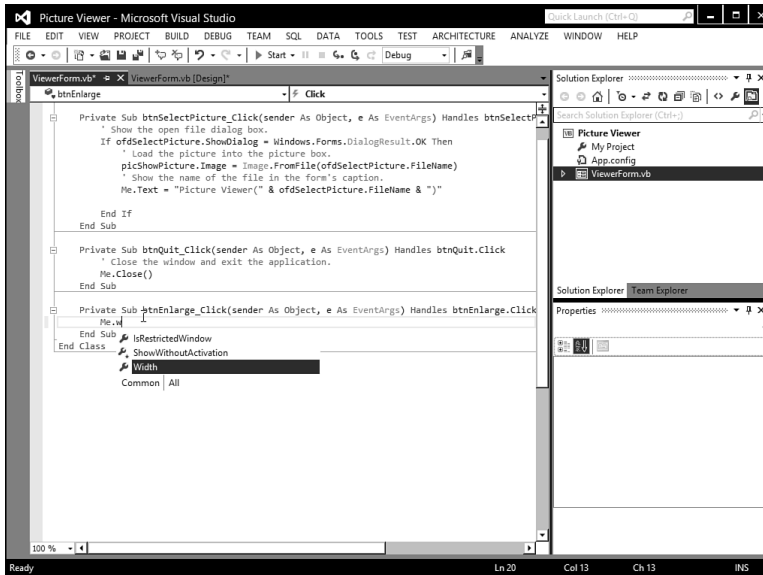


To complete the project, you need to add the small amount of Visual Basic code necessary to modify the form's Height and Width properties when the user clicks a button.

3. Access the code for the Enlarge button by double-clicking the ^ button. Type the following statement exactly as you see it here. Do not press the Enter key or add a space after you've entered this text:

```
Me.Width
```

When you type the period, or *dot*, as it's called, a small drop-down list like the one shown in Figure 3.3 appears. Visual Basic is smart enough to realize that *Me* represents the current form (more on this in a moment). To help you write code for the object, it gives you a drop-down list containing all the properties and methods of the form. This feature is called *IntelliSense*. When an IntelliSense drop-down box appears, you can use the up and down arrow keys to navigate the list and press Tab to select the highlighted list item. This prevents you from misspelling a member name, thereby reducing compile errors. Because Visual Basic is fully object-oriented, you'll come to rely on IntelliSense drop-down lists in a big way; I think I'd rather dig ditches than program without them.



**FIGURE 3.3** IntelliSense drop-down lists (also called auto-completion drop-down lists) make coding dramatically easier.

- Use the Backspace key to erase the code you just entered, and enter the following code in its place (press Enter at the end of each line):

```
Me.Width = Me.Width + 20
Me.Height = Me.Height + 20
```

Remember from before that the word `Me` doesn't refer to a person; it refers to the object to which the code belongs (in this case, the form). `Me` is a *reserved* word; it's a word that you can't use to name objects or variables because Visual Basic has a specific meaning for it. When writing code within a form module, as you're doing here, always use the reserved word `Me` rather than the name of the form. `Me` is much shorter than using the full name of the current form, and it makes the code more portable. (You can copy and paste the code into another form module and not have to change the form name to make the code work.) Also, should you change the name of the form at any time in the future, you won't have to change references to the old name.

The code you just entered does nothing more than set the form's `Width` and `Height` properties to their current value plus 20 pixels.

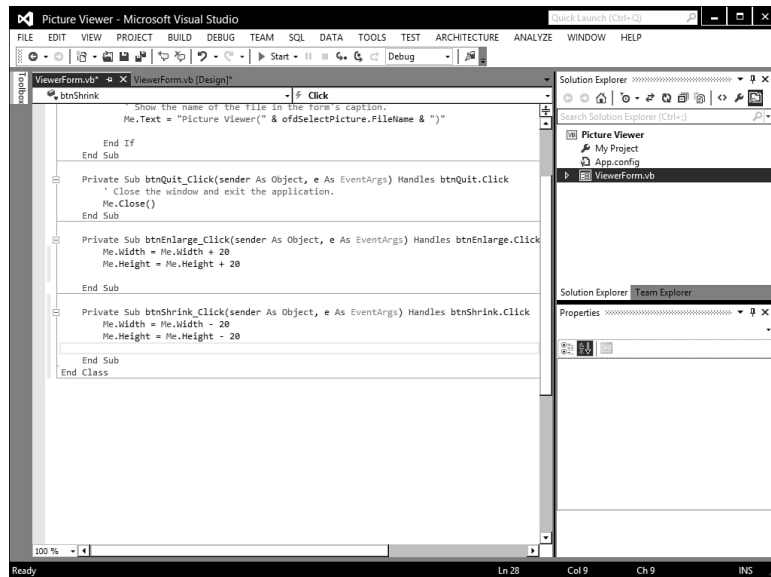
5. Redisplay the form designer by selecting the tab named ViewerForm.vb [Design] at the top of the designer window. Then double-click the button with the `v` to access its `Click` event, and add the following code:

```
Me.Width = Me.Width - 20
Me.Height = Me.Height - 20
```

This code is similar to the code in the `btnEnlarge_Click` event, except that it reduces the form's `Width` and `Height` properties by 20 pixels. Your screen should now look like Figure 3.4.

**FIGURE 3.4**

The code you've entered should look exactly like this.

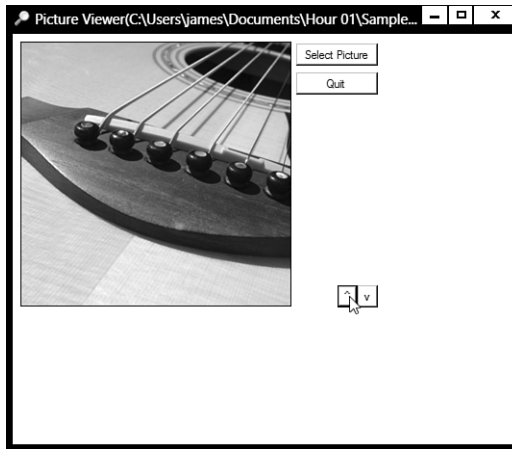


### ***Did you Know?***

As you create projects, it's a good idea to save frequently. When an asterisk appears to the right of a tab's title (as you can see in Figure 3.4), you know that the file edited within that tab has been changed but not saved. Save your project now by clicking the Save All button on the toolbar.

Again, display the form designer by clicking the tab ViewerForm.vb [Design]. Your Properties Example project is now ready to be run! Follow these steps to test your changes:

1. Press F5 to put the project in Run mode.
2. Click the Select Picture button, and choose a picture from your hard drive.
3. Click the `^` button a few times and notice how the form gets bigger (see Figure 3.5).

**FIGURE 3.5**

The form gets bigger, but it still looks just as you designed it.

4. Next, click the v button to make the form smaller. When you've clicked enough to satisfy your curiosity (or until you get bored), end the running program, and return to Design mode by clicking the Stop Debugging button on the toolbar.

Did you notice how the buttons and the image on the form didn't resize as the form's size changed? In Hour 6, "Building Forms: Advanced Techniques," you'll learn how to make your forms resize their contents.

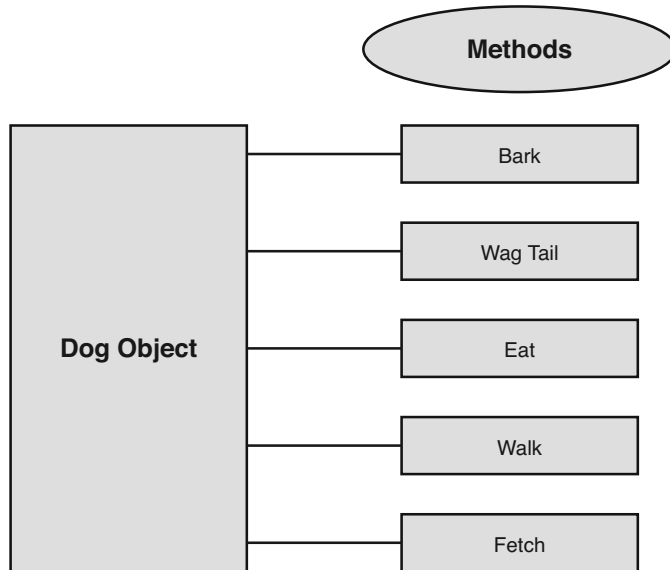
## Understanding Methods

In addition to properties, most objects have *methods*. Methods are actions the object can perform, in contrast to attributes, which describe the object. To understand this distinction, think about the pet object example one more time. A Dog object has a certain set of actions it can perform. These actions, called methods in Visual Basic, include barking, tail wagging, and chewing carpet (don't ask). Figure 3.6 illustrates the Dog object and its methods.



**FIGURE 3.6**

Invoking a method causes the object to perform an action.



## Triggering Methods

Think of methods as functions—which is exactly what they are. When you invoke a method, code is executed. You can pass data to a method, and methods can return values. However, a method is not required to accept parameters (data passed to it by the calling code) or return a value; many methods simply perform an action in code. Invoking (triggering) a method is similar to referencing the value of a property. First you reference the object's name, and then provide a dot, and then the method name: `ObjectName.Method`

For example, to make the hypothetical Dog object Bruno bark using Visual Basic code, you would use this line of code:

```
Bruno.Bark()
```

### **By the Way**

Methods generally are used to have an object perform an action, such as saving or deleting a record in a database. Properties, on the other hand, are used to get and set the object's attributes. One way to tell in code whether a statement is a property reference or method call is that a method call has a set of parentheses after it, like this:

```
AlbumForm.ShowDialog()
```

Invoking methods is simple; the real skill lies in knowing what methods an object supports and when to use a particular method.

## Understanding Method Dynamism

Properties and methods go hand in hand, and at times a particular method might become unavailable because of one or more property values. For example, if you were to set the `NumberOfLegs` property on the Dog object Bruno equal to 0, the `Walk()` and `Fetch()` methods obviously would be inapplicable. If you were to set the `NumberOfLegs` property back to 4, you could then trigger the `Walk()` or `Fetch()` methods again.

## Building a Simple Object Example Project

The only way to really grasp what objects are and how they work is to use them. I've said this before, but I can't say it enough: Everything in Visual Basic 2012 is an object. This fact has its good points and bad points. One of the bad points is that in some instances, it takes more code to accomplish a task than it did in the past—sometimes more characters, sometimes more statements. If you're moving from Visual Basic 6, you have some learning and adjusting ahead of you, but it's worth the effort!

Every project you've built so far uses objects, but now you'll create a sample project that specifically illustrates using objects. If you're new to programming with objects, you'll probably find this a bit confusing. However, I'll walk you through step by step, explaining each section in detail.

You'll modify your Picture Viewer project to include a button that, when clicked, draws a colored border around the picture. You'll get a taste of some drawing functions in this example. Don't worry; you're not expected to understand all the intricacies of the drawing code. Your sole responsibility is grasping how objects work.

## Creating the Interface for the Drawing Project

Continuing with the Picture Viewer project you've been using in this hour, add a new button to the form, and set its properties as follows:

Property	Value
Name	btnDrawBorder
Location	295,70
Size	85,23
Text	Draw Border

## Writing the Object-Based Code

Now you'll add code to the button's `Click` event. I'll explain each statement, and at the end of the steps, I'll show the complete code listing. Follow these steps to create the code that draws the border:

1. Double-click the Draw Border button to access its `Click` event.
2. Enter the first line of code as follows (remember to press `Enter` at the end of each statement):

```
Dim objGraphics As Graphics
```

Here you've just created a variable that will hold an instance of an object. Objects don't materialize out of thin air; they have to be created. When a form is loaded into memory, it loads all its controls (that is, it creates the control objects), but not all objects are created automatically as they are in this situation. The process of creating an instance of an object is called *instantiation*. When you load a form, you instantiate the form object, which in turn instantiates its control objects. You could load a second instance of the form, which in turn would instantiate a new instance of the form and new instances of all controls. You would then have two forms in memory, and two of each used control.

To instantiate an object in code, you create a variable that holds a reference to an instantiated object. The `Dim` statement you wrote in step 2 creates a new variable called `objGraphics`, which holds a reference to an object of type `Graphics`. You'll learn more about variables in Hour 11.

Next, enter the second line of code exactly as shown here:

```
objGraphics = Me.CreateGraphics
```

`CreateGraphics` is a method of the form. (Remember, the keyword `Me` is shorthand for referencing the current form.) Under the hood, the `CreateGraphics` method is pretty complicated. For now, understand that the method `CreateGraphics` instantiates a new object that represents the client area of the current form. The client area is the gray area within a form's borders and title bar. Anything drawn on the `objGraphics` object appears on the form. What you've done is set the variable `objGraphics` to point to an object that was returned by the `CreateGraphics` method. Notice how values returned by a property or method don't have to be traditional values such as numbers or text; they could also be objects.

Enter the third line of code:

```
objGraphics.Clear(SystemColors.Control)
```

This statement clears the form's background using whatever color the user has selected as the Windows Control color, which Windows uses to paint forms.

How does this happen? After declaring the `objGraphics` object, you used the `CreateGraphics` method of the form to instantiate a new graphics object in the variable `objGraphics`. With the code statement you just entered, you're calling the `Clear()` method of the `objGraphics` object. The `Clear()` method is a method of all Graphics objects used to clear the graphic surface. The `Clear()` method accepts a single parameter: the color you want used to clear the surface.

The value you're passing to the parameter might seem a bit odd. Remember that "dots" are a way of separating objects from their properties and methods. (Properties, methods, and events are often called object *members*.) Knowing this, you can discern that `SystemColors` is an object because it appears before any of the dots. Object references can and do go pretty deep, and you'll use many dots throughout your code. The key points to remember are

- ▶ Text that appears to the left of a dot is always an object (or namespace).
- ▶ Text that appears only to the right of a dot is a property reference or method call. If the text is followed by parentheses, it's a method call. If not, it's most likely a property.
- ▶ Methods can return objects, just as properties can. The only surefire ways to know whether the text between two dots is a property or method are to look at the icon of the member in the IntelliSense drop-down or to consult the object's documentation.

The final text in this statement is the word `Control`. Because `Control` isn't followed by a dot, you know that it's not an object; therefore, it must be a property or method. You expect this string of object references to return a color value to be used to clear the graphic object. Therefore, you know that `Control` in this instance must be a property or method that returns a value (because you need the return value to set the `Clear()` method). A quick check of the documentation would tell you that `Control` is indeed a property. The value of `Control` always equates to the color designated on the user's computer for the face of forms and buttons. By default, this is a light gray (often fondly referred to as *battleship gray*), but users can change this value on their computers. By using this property to specify a color rather than supplying the actual value for gray, you're assured that no matter the color scheme used on a computer, the code will clear the form to the proper system color.

Enter the following statement. (Note: Do not press Enter until you're finished entering *all* the code shown here. The code appears on two lines only because of the size restriction of this page.)

```
objGraphics.DrawRectangle(Pens.Blue, picShowPicture.Left - 1,  
picShowPicture.Top - 1, picShowPicture.Width + 1, picShowPicture.Height + 1)
```

This statement draws a blue rectangle around the picture on the form. Within this statement is a single method call and five property references. Can you tell what's what? Immediately following `objGraphics` (and a dot) is `DrawRectangle`. Because no equals sign is present but there is an open parenthesis, you can deduce that this is a method call. As with the `Clear()` method, the parentheses after `DrawRectangle` are used to enclose values passed to the method.

The `DrawRectangle()` method accepts the following parameters in the order in which they appear here:

- ▶ A pen
- ▶ X value of the upper-left corner
- ▶ Y value of the upper-left corner
- ▶ Width of the rectangle
- ▶ Height of the rectangle

The `DrawRectangle()` method draws a perfect rectangle using the X, Y, Width, and Height values passed to it. The attributes of the line (color, width, and so on) are determined by the pen specified in the Pen parameter. (I won't go into detail on pens here; check the online help if pens interest you.) Looking at the dots once more, notice that you're passing the `Blue` property of the `Pens` object. `Blue` is an object property that returns a predefined Pen object that has a width of 1 pixel and the color blue.

For the next two parameters, you pass property values. Specifically, you pass the top and left values for the picture, less 1. If you passed the exact left and top values, the rectangle would be drawn on the form at exactly the top and left properties of the `PictureBox`. You wouldn't see them because controls by default overlap any drawing performed on the form.

The last two property references are for the height and width of the `PictureBox`. Again, we adjust the values by 1 to ensure that the rectangle is drawn outside the borders of the `PictureBox`.

Finally, you have to clean up after yourself by entering the following code statement:

```
objGraphics.Dispose()
```

Objects often use other objects and resources. The underlying mechanics of an object can be mind-boggling and are almost impossible to discuss in an entry-level programming book. The net effect, however, is that you must explicitly destroy most objects when you're finished with them. If you don't destroy an object, it might persist in memory, and it might hold references to other objects or resources that exist in memory. This means that you can create a *memory leak* within your application that slowly (or rather quickly) munches system memory and resources. This is one of the cardinal no-no's of Windows programming. However, the nature of using resources and the fact that you're responsible for telling your objects to clean up after themselves make this easy to do. If your application causes memory leaks, your users won't call for a plumber, but they might reach for a monkey wrench—in an effort to smack you upside the head!

Objects that must explicitly be told to clean up after themselves usually provide a `Dispose()` method. When you're finished with such an object, call `Dispose()` on the object to make sure that it frees any resources it might be holding.

For your convenience, here are all the lines of code:

```
Dim objGraphics As Graphics
objGraphics = Me.CreateGraphics
objGraphics.Clear(System.Drawing.SystemColors.Control)

objGraphics.DrawRectangle(System.Drawing.Pens.Blue, _
    picShowPicture.Left - 1, picShowPicture.Top - 1, _
    picShowPicture.Width + 1, picShowPicture.Height + 1)

objGraphics.Dispose()
```

The statement calling `DrawRectangle()` is shown here as three lines of code. At the end of the first and second lines is an underscore character (`_`), also known as a *line continuation character*. It tells the Visual Basic compiler that the statement immediately following the character is a continuation of the current statement. You can, and should, use this character to break up long statements in your code.

**By the  
Way**

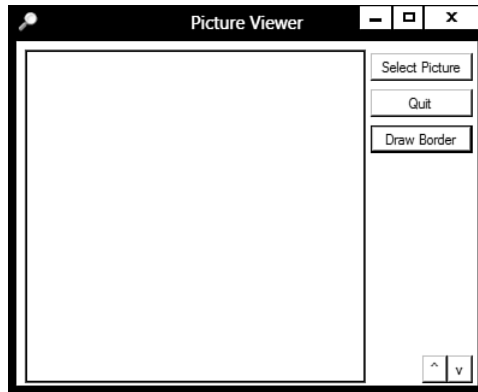
Click **Save All** on the toolbar to save your work before continuing.

## Testing Your Object Example Project

Now the easy part: Run the project by pressing `F5` or by clicking the **Start** button on the toolbar. Your form looks pretty much as it did at design time. Clicking the button causes a blue rectangle to be drawn around the `PictureBox`, as shown in Figure 3.7.

**FIGURE 3.7**

You can create simple lines and complex drawings by using objects.



### **By the Way**

If you receive any errors when you attempt to run the project, go back and make sure that the code you entered exactly matches the code I've provided.

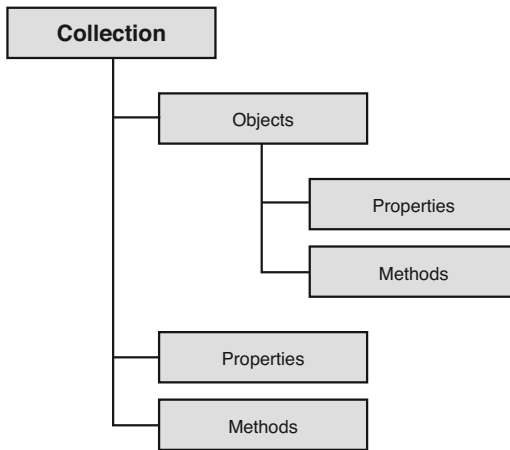
If you use Alt+Tab to switch to another application after drawing the rectangle, the rectangle will be gone when you come back to your form. In fact, this occurs any time you overlay the graphics with another form. In Hour 18, "Working with Graphics," you will learn how to persist images so that they don't disappear when the form becomes obscured.

Stop the project now by clicking Stop Debugging on the Visual Basic toolbar. What I hope you've gained from building this example is not necessarily that you can now draw a rectangle (which is cool), but rather an understanding of how objects are used in programming. As with learning almost anything, repetition aids understanding. That said, you'll be working with objects *a lot* throughout this book.

## Understanding Collections

A *collection* is just what its name implies: a collection of objects. Collections make it easy to work with large numbers of similar objects by enabling you to create code that performs iterative processing on items within the collection. *Iterative processing* is an operation that uses a loop to perform actions on multiple objects, rather than requiring you to write the operative code for each object. In addition to containing an indexed set of objects, collections have properties and might have methods. Figure 3.8 illustrates the structure of a collection.

Continuing with the Dog/Pet object metaphor, think about what an `Animals` collection might look like. The `Animals` collection might contain one or more pet objects, or it might be empty (contain no objects). All collections have a `Count` property that returns the total count of objects contained in the collection. Collections might also have methods, such as a `Delete()` method used to remove objects from the collection, and an `Add()` method used to add a new object to the collection.

**FIGURE 3.8**

Collections contain sets of like objects, and they have their own properties and methods.

To better understand collections, you'll create a small Visual Basic project that cycles through the `Controls` collection of a form and tells you the value of the `Name` property of every control on the form. To create your sample project, follow these steps:

1. Start Visual Basic (if it's not already loaded), and create a new Windows Application project titled **Collections Example**.
2. Change the form's filename to `CollectionsExampleForm.vb` using the Solution Explorer (right-click `Form1.vb` in the Solution Explorer and choose `Rename`), and set the form's `Text` property to **Collections Example** in the Properties window (you need to click the form first to view its properties).
3. Add a new button to the form by double-clicking the `Button` tool in the toolbox. Set the button's properties as follows:

Property	Value
Name	<code>btnShowNames</code>
Location	<code>88, 112</code>
Size	<code>120, 23</code>
Text	<code>Show Control Names</code>

4. Next, add some `TextBox` and `Label` controls to the form. As you add the controls to the form, be sure to give each control a unique name. Feel free to use any name you want, but you can't use spaces in a control name. You might want to drag the controls to different locations on the form so that they don't overlap.



5. When you're finished adding controls to your form, double-click the Show Control Names button to add code to its Click event. Enter the following code:

```
Dim intIndex As Integer
For intIndex = 0 To Me.Controls.Count - 1
    MessageBox.Show("Control #" & intIndex & " has the name " & _
        Me.Controls(intIndex).Name)
Next intIndex
```

### **By the Way**

Every form has a Controls collection, which might not contain any controls; even if no controls are on the form, the form still has a Controls collection.

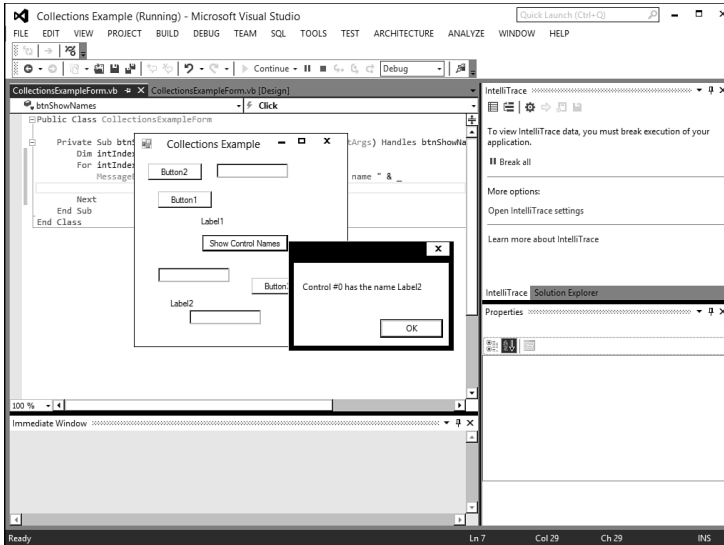
The first statement of the preceding code should look familiar to you by now. As with the Object example you created earlier, this statement creates a variable to hold a value. Rather than create a variable that can hold an object, as you did in the earlier example, this statement creates a variable that can hold only a number.

The next statement (the one that begins with For) accomplishes a few tasks. First, it initializes the variable intIndex to 0, and then it starts a loop (loops are discussed in Hour 14, "Looping for Efficiency"). It increments intIndex by 1 until intIndex equals the number of controls on the form, less 1. The reason you subtract 1 from the Count property is that collections are 0-based—the first item is always item 0. Thus, the first item is in index 0, the second item is in location 1, and so forth. If you tried to reference an item of a collection in the location of the value of the Count property, an error would occur. You would be referencing an index that's 1 higher than the actual locations within the collection.

The MessageBox.Show() method (mentioned in Hour 2 and discussed in detail in Hour 17, "Interacting with Users") is a class of the .NET Framework that's used to display simple dialog boxes with text. The text that you're providing, which the Show() method displays, is a concatenation of multiple strings of text. (*Concatenation* is the process of adding strings together; it's discussed in Hour 12, "Performing Arithmetic, String Manipulation, and Date/Time Adjustments.")

Run the project by pressing F5 or by clicking Start on the toolbar. Ignore the additional controls that you placed on the form, and click the Show Control Names button. Your program then displays a message box similar to the one shown in Figure 3.9 for each control on your form (because of the loop). When the program is finished displaying the names of the controls, choose Debug, Stop Debugging to stop the program, and then save the project.

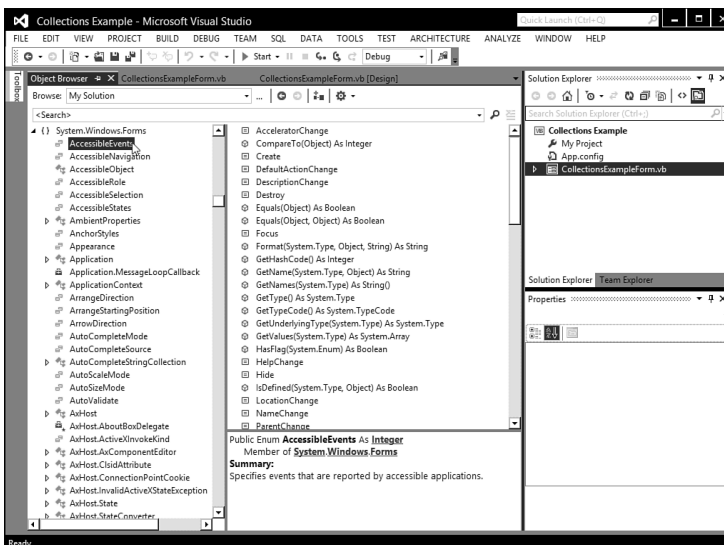
Because everything in Visual Basic 2012 is an object, you can expect to use numerous collections as you create your programs. Collections are powerful, and the quicker you become comfortable using them, the more productive you'll be.



**FIGURE 3.9**  
The Controls collection enables you to get to every control on a form.

## Using the Object Browser

Visual Basic 2012 includes a useful tool that enables you to easily view members (properties, methods, and events) of all the objects in a project: the Object Browser (see Figure 3.10). This is useful when you're dealing with objects that aren't well documented because it enables you to see all the members an object supports. To view the Object Browser, press F2 or select it in the View menu.



**FIGURE 3.10**  
The Object Browser enables you to view all properties and methods of an object.

The Browse drop-down list in the upper-left corner of the Object Browser is used to determine the *browsing scope*. You can choose My Solution to view only the objects referenced in the active solution, or you can choose All Components to view all possible objects. You can customize the object set by clicking the drop-down arrow next to the Object Browser Settings button to the far right of the Browse drop-down list. I don't recommend changing the custom object setting until you have some experience using Visual Basic .NET objects, as well as experience using the Object Browser.

The top-level nodes (each item in the tree is called a *node*) in the Objects tree are libraries. *Libraries* are usually DLL or EXE files on your computer that contain one or more objects. To view the objects in a library, simply expand the library node. As you select objects within a library, the list to the right of the Objects tree shows information about the members of the selected object (refer to Figure 3.10). For even more detailed information, click a member in the list on the right. The Object Browser shows information about the member in the area below the member list.

## Summary

In this hour, you learned a lot about objects. You learned how objects have properties, which are attributes that describe the object. Some properties can be set at design time in the Properties window, and most can also be set at runtime in Visual Basic code. You learned that referencing a property on the left side of the equals sign has the effect of changing the property, whereas referencing a property on the right side of the equals sign retrieves the property's value.

In addition to properties, you learned that objects have executable functions, called methods. Like properties, methods are referenced by using a dot at the end of an object reference. An object might contain many methods and properties, and some properties can even be objects themselves. You learned how to “follow the dots” to interpret a lengthy object reference.

Objects are often used as a group, called a collection. You learned that a collection often contains properties and methods, and that collections let you easily iterate through a set of like objects. Finally, you learned that the Object Browser can be used to explore all the members of an object in a project.

The knowledge you've gained in this hour is fundamental to understanding programming with Visual Basic, because objects and collections are the basis on which applications are built. After you have a strong grasp of objects and collections—and you will have by the time you've completed all the hours in this book—you'll be well on your way to fully understanding the complexities of creating robust applications with Visual Basic 2012.

## Q&A

**Q.** *Is there an easy way to get help about an object's member?*

**A.** Absolutely. Visual Basic's context-sensitive Help extends to code as well as visual objects. To get help on a member, write a code statement that includes the member (it doesn't have to be a complete statement), position the cursor within the member text, and press F1. For instance, to get help on the Integer data type, you could type Integer, position the cursor within the word Integer, and press F1.

**Q.** *Are there any other types of object members besides properties and methods?*

**A.** Yes. An event is actually a member of an object, although it's not always thought of that way. Although not all objects support events, most objects do support properties and methods.

## Workshop

### Quiz

1. True or false: Visual Basic 2012 is a true object-oriented language.
2. An attribute that defines the state of an object is called a \_\_\_\_\_.
3. For you to change the value of a property, the property must be referenced on which side of the equals sign?
4. What is the term for when a new object is created from a template class?
5. An external function of an object (one that is available to code manipulating an object) is called a \_\_\_\_\_.
6. True or false: A property of an object can be another object.
7. A group of like objects is called a \_\_\_\_\_.
8. What tool is used to explore the members of an object?

## Answers

1. True
2. Property
3. The left side
4. Instantiation
5. Method
6. True
7. Collection
8. The Object Browser

## Exercises

1. Create a new project, and add two text boxes and a button to the form. Write code that, when a button is clicked, places the text in the first text box into the second text box. Hint: Use the Text property of the TextBox controls.
2. Modify the collections example in this hour to print the height of all controls, rather than the name.

# Index

## A

- accelerator keys, 198
- accept buttons, 154-156
- AcceptButton property, 154-155
- AcceptsReturn property, 150
- ActiveCaption system color, 394
- ActiveCaptionText system color, 394
- ActiveCell objects, 474
- ActiveX controls. *See* user controls
- Add() method, 72
- addition operations, 270
- ADO.NET, 452
  - ConnectionString property, 454-455
  - DataAdapter objects, 452
    - creating, 456-457
  - databases
    - closing data source connections, 455
    - connecting to, 453-455
  - DataRows, 459
  - 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
    - SqlConnection objects, 452-453
    - SqlDataAdapter objects, creating, 456-457
    - System.Data namespace, 452
- aligning
  - controls, 125-126
  - text in text boxes, 148
- anchoring controls, 128-131
- And operator, 276
- applications
  - automating
    - Excel (MS), 470-476
    - Word (MS), 476-477
  - distributed applications, uninstalling, 486
  - publishing with Publish Wizard (ClickOnce technology), 482-484, 488-489
  - testing, 486
- arithmetic operations, 270
  - 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

### automation

- defining, 469
- Excel (MS), 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
- system requirements, 478
- Word (MS), 476
  - instances of automation server creation, 477
  - library reference creation, 476-477

### autosizing controls, 128-131

## B

- BackColor** property, 40-41, 98-100
- 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, creating Graphics objects for, 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, 40-41
- break points, 328-330**
- browsing files, writing code for, 22-24**
- build errors, 326-327**
- Button controls, 81**
- buttons**
  - accept buttons, 154-156
  - Buttons collection, 208-210
  - cancel buttons, 154-156
  - check boxes, 156
  - creating, 153
  - radio buttons, 159-160
  - toolbar buttons, drop-down menus, 212
  - user messages
    - determining which button is clicked, 372
    - specifying buttons in, 369-370
- Byte data type, 239**

## C

- cancel buttons, 154-156
- CancelButton** property, 155-156
- Case Else** statements, 303
- Case** statements, 300, 303
- casting data between data types, 241-242**
- cells (Excel), manipulating in workbooks, 473-475**
- Char** data type, 239
- check boxes, 156
- checked menu items, creating, 200-201**
- CheckState** property, 156
- circles, drawing, 398**
- class modules, 218**
  - defining, 48
- 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, 357
    - 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 event handler, 24-25
- Click events, 81, 84, 153-155, 382, 477
- ClickOnce technology, 481-482
  - Publish Wizard, 482-484, 488-489
- clients, defining, 469
- Clng() function, 331-333
- Close() method, 455
- closing
  - For...Next loops, 310
  - forms, 112-113
  - windows, 10
- CLR (Common Language Runtime), 492
- code
  - comments, adding, 324-325
  - debugging, 323-324
    - adding comments to code, 324-325
  - break points, 328-330
  - build errors, 326-327
  - creating error handlers via Try...Catch...Finally structures, 336-338
  - handling exceptions in error handlers, 339-342
  - Immediate window, 331-335
  - On Error statements, 336
  - runtime errors, 326-327
  - encapsulating via classes, 348
  - writing, 21
    - browsing files example, 22-24
    - terminating programs example, 24-25
- code labels, 304-305
- code modules
  - class modules, 218
  - defining, 217
- code procedures
  - calling, 225-228
    - passing parameters, 228-230
  - creating, 219-220
    - declaring procedures with return values, 224-225
    - declaring procedures without return values, 220-224
  - exiting, 231-232
  - infinite recursion, 232
- collections, 73-74. *See also* objects
- color
  - Buttons collection, 208-210
  - Controls collections, 73-74
  - iterative processing, 72
- color
  - color properties, 42-44
  - forms, changing background color, 98-99
  - system colors, 393-396
- columns
  - DataRows, 459
  - enhanced lists, creating columns for, 183
- COM components, 472
- combo boxes, 161
  - drop-down lists, creating, 168-170
- CommandBuilder objects, 457
- comments, adding to code, 324-325
- common language runtime (CLR), 492
- 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, panels, 157-158
- context menus, 204-206
  - Context Menu Strip control, 205
- Continue Do statements, Do...Loop loops, 317
- Continue For statements, For...Next loops, 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, 128-131
  - autosizing, 128-131
  - buttons
    - accept buttons, 154-156
    - cancel buttons, 154-156
    - check boxes, 156
    - creating, 153
    - radio buttons, 159-160
  - check boxes, 156



**controls**

- combo boxes, 161
  - creating drop-down lists, 168-170
- containers
  - group boxes, 157-158
  - panels, 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, 16-21, 38-39, 118
  - double-clicking controls in toolbox, 118
  - dragging controls from toolbox, 118
  - drawing controls, 119
- forms, organizing on
  - aligning controls, 125-126
  - anchoring controls, 128-131
  - autosizing controls, 128-131
  - creating tab orders, 132-134
  - grid (size and snap), 119-121
  - layering controls, 134
  - selecting groups of controls, 123-125
  - setting property values for groups of controls, 127
  - sizing controls, 126
  - Snap to Lines, 122
  - spacing groups of controls, 127
- Graphics objects, creating for controls, 390
- Group Box control, 157-158
- group boxes, 157-158
- groups of controls
  - setting property values, 127
  - spacing, 127
- hierarchical lists, creating, 187
  - adding nodes, 188-189
  - clearing all nodes, 190
  - removing nodes, 190
- Image List control, storing images in, 180-181
- invisible-at-runtime controls, adding to forms, 20-21
- Label control, displaying static text, 145-147
- layering, 134
- 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
- List View control, 182-187
- Menu Strip control, 196-198
- nonvisual controls, adding to forms, 20-21
- OpenFileDialog control, 17, 20, 23, 26, 409-412, 415
  - creating file filters, 412
  - displaying Open File dialog, 413
- Panel control, 157-158
- panels, 157-158
- Picturebox controls, 17-19
- SaveFileDialog control, 409, 413-415
- selecting groups of controls, 123-125
- sizing, 126
  - autosizing, 128-131
- Status Bar control, 213-214
- Tab control, 177-180
- tab orders, creating, 132-134
- 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, 17-18

**Controls collections, 73-74**

**ControlText system color, 395**

**Copy() method, 417**

**copying files, 416-417**

**CreateDirectory() method, 424**

**CreateGraphics method, 68**

**CreateSubKey() method, 431**

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

**customizing Visual Basic, 32**

docking design windows, 34-35

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**

creating, 456-457

**databases, 451-452**

ADO.NET, 452

closing data source  
connections, 455

DataAdapter objects, 452,  
456-457

database connections,  
453-455

DataRows, 459

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

DataRows, referencing fields  
in, 459

deleting records, 464

editing records, 462

navigating records, 460-462

populating, 458

**Date data type, 239-240, 283-284**

**date/time**

adding time to specific dates,  
285-286

current system date/time,  
retrieving, 288

Date data type, **239-240,**  
283-284

DateAdd() function, 285-286

DateDiff() function, 286-287

DatePart() function, 287

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, 335**

**debugging code, 323-324**

break points, 328-330

build errors, 326-327

comments, adding to code,  
324-325

**debugging code**

- error handlers
  - creating via
    - Try...Catch...Finally structures, 336-338
    - handling exceptions, 339-342
  - Immediate window, 331-335
  - On Error statements, 336
  - Registry deployments (Windows), 437
  - runtime errors, 326-327
- Decimal data type, 239-240**
- declaration statements (events), 84**
- declaring**
  - arrays, 251
  - objects, creating new objects
    - when dimensioning variables, 360-361
  - variables, 244-246
    - explicit variable declaration, 247-248
    - static scope, 258-259
- Delete() method, 72, 418-419, 424**
- DeleteSubKey() method, 431**
- deleting**
  - DataTable object records, 464
  - directories (folders), 424
  - files, 418-419
  - menu items, 200
  - objects, .NET Framework, 496
  - Registry keys (Windows), 431
- design windows**
  - displaying, 33
  - docking, 34-35
  - floating, 34
  - hiding, 33, 36
- Desktop system color, 395**
- dialog boxes, creating**
  - custom dialog boxes, 373-375
  - tabbed dialog boxes, 177-180
- dimensioning**
  - arrays, 251
  - objects, creating new objects
    - when dimensioning variables, 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, 369-370
- Open File dialog, 413
- properties (objects), 11
- Registry options (Windows), 434
- static text via Label control, 145-147
- text files, 445-447
- toolbars, 37
- windows, Visual Studio 2010, 10
- Dispose() method, 71**
- distributable components, defining, 6**
- distributed applications, uninstalling, 486**
- 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, 34-35**
- Double data type, 239-240**
- double-clicking in Visual Studio 2010, 10**
- 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

drop-down menus, toolbar  
buttons, 212

## E

early binding, 358

object variables, 360

editing `DataTable` object  
records, 462

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,  
295-296

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` struc-  
tures, 336-338

exceptions, handling, 339-342

event handlers, 22

Click event handler, 24-25

creating, 89-92

events, 22

build example, 87

event handler creation,  
89-92

user interface creation,  
87-88

Click events, 81, 84, 153-  
155, 382, 477

declaration statements, 84

event handlers, 22

Click event handler, 24-25

creating, 89-92

event-driven programming,  
defining, 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

object events, accessing,  
83-85

`Paint` events, 82, 405

parameters, 84-87

defining, 85

multiple parameters in  
events, 85

recursive events, 82

Resize events, 405

`TextChanged` events, 81, 153

Excel (MS) automation, 470

cell manipulation in work-  
books, 473-475

forcing showing of Excel, 473

instances of automation serv-  
er 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, Else state-  
ments, 296

file filters, creating, 412

files

attribute flags, 421

attributes, retrieving, 420-421

browsing, writing code for,  
22-24

copying, 416-417

date/time information, retriev-  
ing, 420

deleting, 418-419

existence of, determining, 416

## files

- File object, 415-423
- log files
  - creating, 443-444
  - displaying, 445-447
  - testing, 447
- moving, 417-418
- naming, 418
- properties, retrieving, 421-423
- text files
  - creating, 443-444
  - displaying, 445-447
  - reading, 441-442
  - testing, 447
  - writing, 439-441
- Filter property, 21**
- filters, creating file filters, 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, 41**
- For statements, 310**
- 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
- form objects, 58**
- formatting date/time, 287-288**
- FormBorderStyle property, 105-107**
- FormClosed events, 455**
- forms**
  - backgrounds
    - adding images to, 100-101
    - changing color, 98-99
  - borders, customizing, 105-107
  - closing, 112-113
  - control box buttons, adding to forms, 103-104
  - controls, adding to, 16-21, 38-39, 118
    - double-clicking controls in toolbox, 118
    - dragging controls from toolbox, 118
    - drawing controls, 119
  - controls, organizing
    - aligning controls, 125-126
    - anchoring controls, 128-131
    - autosizing controls, 128-131
    - creating tab orders, 132-134
    - grid (size and snap), 119-121
    - layering controls, 134
    - selecting groups of controls, 123-125
    - setting property values for groups of controls, 127
    - sizing controls, 126
    - Snap to Lines, 122
    - spacing groups of controls, 127
  - defining, 9, 48, 95
  - displaying in
    - maximized state, 109-110
    - minimized state, 109-110
    - normal state, 109-110
    - specific initial position, 111
  - Graphics objects, creating for forms, 390
  - graphics, persisting on forms, 400
  - icons, assigning to forms, 14, 103
  - maximize buttons, adding to forms, 103-104
  - MDI (multiple-document interface) forms, 137-140
  - menus
    - 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, 196-198
    - creating menu items, 199
    - deleting menu items, 200

## If...Then constructs

- moving menu items, 200
- programming, 202-204
- status bars, 213-214
- toolbars, 208-212
- minimize buttons, adding to forms, 103-104
- modality, 108
- naming, 96
- nonmodal windows, 134
- scrollable forms, 135-136
- showing, 107-108
- sizing, 15, 107-110
- startup forms, configuring, 140-141
- taskbar, preventing forms from appearing in, 112
- text bars, displaying text on, 97-98
- title bar, changing text in, 13
- topmost nonmodal windows, 134
- transparent forms, 135
- unloading, 112-113
- Windows Forms Applications, creating, 8-9

**FormulaR1C1 property, 474**

functions, exposing as methods, 356

**G**

Get construct, 354

GetAttributes() method, 420-421

GetValue() method, 432

global (namespace) scope, 256-257

GoTo statements, 304-306

graphics. *See also* images

- circles, drawing, 398
- Clear() method, 69
- CreateGraphics method, 68
- 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
  - creating, 390-391
- 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, 119-121
- ShowGrid property, 120-121
- SnapToGrid property, 120-121

**Group Box control, 157-158**

**group boxes, 157-158**

**grouping controls, 123-125**

**H**

handlers (event), creating (event building example), 89-92

Height property, 15

help
 

- finding, 53
- further reading, 497

**hiding**

- design windows, 33, 36
- Excel (MS) automation, 473
- toolbars, 37
- windows, Visual Studio 2010, 10

**hierarchical lists, creating, 187**

- nodes
  - adding, 188-189
  - clearing all nodes, 190
  - removing, 190

**Highlight system color, 395**

**HighlightText system color, 395**

**hives, Registry (Windows), 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-J****icons**

- forms, assigning icons to, 14, 103
- user messages, specifying icons in, 369-370

**IDE (integrated development environment), 7**

**If statements, 274**

**If...End constructs, 435**

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

- Else statements, 296

- Elseif statements, 297-298

**If...Then constructs**

- End If statements, 295
  - expressions
    - Elseif statements, 299
    - evaluating expressions for multiple values, 298
    - executing False expressions, 296
  - IsNumeric() function, 294
  - nesting, 297-298
  - IL (Intermediate Language), 493-494**
  - images. See also graphics**
    - forms, adding images to backgrounds, 100-101
    - Image List control, storing images in, 180-181
  - Immediate window, debugging code, 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**
  - installing**
    - applications, uninstalling distributed applications, 486
    - installation (setup) programs
      - Publish Wizard (ClickOnce technology), 482-484, 488-489
      - testing, 486
  - instantiating objects, 68, 357**
    - binding references to variables, 357-361
    - releasing object references, 362
  - 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, 369-370
    - mouse events, 382-385
    - user information, obtaining, 377-379
  - interfaces, code, writing for, 21**
    - browsing files example, 22-24
    - terminating programs example, 24-25
  - Invalidate() method, 405**
  - invisible-at-runtime controls, adding to forms, 20-21**
  - invoking (triggering)**
    - events, 80
    - objects, 81
    - OS (operating systems), 82
    - user interaction, 81
    - methods (objects), 66
  - IsDate() function, 289**
  - IsNumeric() function, 294**
  - iterative processing, 72**
- K-L**
- keyboards, user/program interaction, 379-382**
  - Label control, displaying static text, 145-147**
  - labels (code), 304-305**
  - late binding object variables, 358-359**
  - layering controls, 134**
  - LayoutMode property, 120**
  - Len() function, 279**
  - libraries**
    - defining, 76
    - type (object) libraries, 470
  - line continuation characters, 71**
  - lines, drawing, 397**
  - list boxes, 161**
    - design time, manipulating items at, 162
    - lists
      - adding items to, 163
      - clearing, 165-166
      - removing items from, 164-165
      - retrieving information from selected items, 166-167
      - sorting, 167-168
    - runtime, manipulating items at, 162-168
  - lists**
    - 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
- in list boxes
  - adding items to, 163
  - clearing, 165-166
  - removing items from, 164-165
  - retrieving information from selected items, 166-167
  - sorting, 167-168
- List View control, 182-187
- Tree View control, 187-190
- literal values, passing to variables, 246**
- local (procedure-level) scope, 255**
- log files**
  - creating, 443-444
  - displaying, 445-447
  - testing, 447
- Long data type, 239-240**
- loops, 315**
  - 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 values, 311
- Step statements, 311
- iterative processing, 72
- recursive loops, 232
- While...End loops, 320

## M

**magic numbers, 242**

**managing projects, Solution Explorer, 45-46**

**math operations, 270**

- 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, 137-140**

**Me.Close() statements, 25**

**memory leaks, 71**

**Menu system color, 395**

**menus**

context menus, 204-206

creating, 196-198

drop-down menus, toolbar buttons, 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, 208

adding buttons to toolbars, 208-210

button drop-down menus, 212

programming, 211

**MenuText system color, 395**

**MessageBox.Show() function, 52, 74, 302, 367-373**

**messages, displaying, 367-368**

buttons

determining which button is clicked, 372

guidelines for creating good messages, 373

specifying, 369-370

icons, specifying, 369-370



**metadata****metadata, 496****methods (objects)**

defining, 65

dynamism, 67

events versus, 80

functions, exposing as  
methods, 356

invoking (triggering), 66

**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, 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 res-  
olution requirements, 10****mouse**

Click events, 382

MouseDown events, 84-86,  
153, 382

MouseEnter events, 382

MouseHover events, 382

MouseLeave events, 382

MouseMove events, 90,  
153, 382MouseUp events, 153, 382  
user/program interaction,  
382-385Visual Studio 2010, double-  
clicking in, 10**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),  
494-495****System namespace, 494****System.Data namespace, 452****naming**

constants, 246

data types, 259-260

files, 418

forms, 96

naming collisions, 494

objects, 11-13, 260

projects, 8

**scope**

name conflicts, 257

naming conventions, 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

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, hierarchical lists**

adding nodes to, 188-189

clearing all nodes, 190

removing nodes from, 190

**nonmodal windows, topmost in  
forms, 134****nonvisual controls, adding to  
forms, 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, defining, 469**

**objects**

ActiveCell objects, 474

building example, 67-71

classifying, 11

CommandBuilder objects, 457

control objects, 58

creating via classes, 350-351

exposing object attributes  
as properties, 352-356

DataAdapter objects, 452

creating, 456-457

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

declaring, creating new  
objects when dimensioning  
variables, 360-361

defining, 11, 57-58

dimensioning, creating new  
objects when dimensioning  
variables, 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

creating, 390-391

instantiating, 68, 357

binding references to vari-  
ables, 357-361

releasing object refer-  
ences, 362

iterative processing, 72

libraries, defining, 76

lifetime of, 363

**methods**

defining, 65

dynamism, 67

exposing functions as  
methods, 356

invoking (triggering), 66

My.Computer.Registry object,  
430-433

naming, 11-13, 260

.NET Framework, deleting  
objects, 496

Object Browser, 75-76

OOP (object-oriented program-  
ming). See classes

prefixes, 260

properties, 11, 39

adding to classes, 352-353

button creation example,  
61-64

changing, 40-41

defining, 58

displaying, 11

read-only properties, 60,  
355-356

readable properties,

creating, 354

referencing, 59-60

unchangeable  
properties, 60

viewing, 40

writable properties, creat-  
ing, 354-355

write-only properties,  
creating, 355-356

Range objects, 474

scope, browsing, 76

selecting, 40

SqlConnection objects,  
452-453

SqlDataAdapter objects, cre-  
ating, 456-457

Startup object, 140-141

StreamReader object, 441-442

StreamWriter object, 439-441  
testing example, 71-72

**Office (MS)**

Excel automation, 470

cell manipulation in work-  
books, 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

Word automation, 476-477

**OLE controls. See user controls**

**On Error statements, 336**

## OOP (object-oriented programming) classes

### OOP (object-oriented programming) classes, 348

- creating objects, 350-356
- encapsulating code/data, 348
- instantiating objects, 357-362
- standard modules versus, 349

### Opacity property, 135

### Open File dialog, displaying, 413

### OpenFileDialog control, 17, 20, 23, 26, 409-412, 415

- file filters, creating, 412
- Open File dialog, displaying, 413

### opening existing projects, 32

### OpenPicture() function, 443

### operator precedence (arithmetic operations), 272-273

### Or operator, 276-277

### OS (operating systems), invoking events, 82

## P

### Paint events, 82, 405

### Panel control, 157-158

### panels, 157-158

### parameters (events), 84-87

- defining, 85
- multiple parameters in events, 85
- passing in code procedures, 228-230

### password fields, creating, 152

### PasswordChar property, 152

### pens, 392-393

### peripherals

- keyboards, user/program interaction, 379-382

- mouse, user/program interaction, 382-385

### PictureBox control, 17-19

### pictures. *See* images

### prefixes

- data types, 259-260
- objects, 260
- scope, 260

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

### procedures, 51

- calling, 225-228
  - passing parameters, 228-230
- 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, 369-370

### mouse events, 382-385

- user information, obtaining, 377-379

### programming

- event-driven programming, defining, 79
- menus, 202-204
- MessageBox.Show() statements, 52
- procedures, 51
- toolbars, 211
- variables, storing values in, 51

### programs

- defining, 47
- projects versus, 47
- running, 25-27
- terminating, writing code for, 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, Solution Explorer, 45-46
- naming, 8
- New Project dialog, 7-8, 30
- opening existing projects, 32
- programs versus, 47
- properties, setting, 48-49
- Recent Projects dialog, 30
- saving, 13

### properties (objects), 39

- AcceptButton property, 154-155

AcceptsReturn property, 150  
 Auto Hide property, 36  
 BackColor property, 40-41, 98-100  
 BackgroundImage property, 100, 102  
 BorderStyle property, 40-41  
 button creation example, 61-64  
 CancelButton property, 155-156  
 changing, 40-41  
 CheckState property, 156  
 classes, adding to, 352-353  
 color properties, 42-44  
 ConnectionString property, 454-455  
 defining, 11, 58  
 descriptions, viewing, 44  
 displaying, 11  
 Enabled property, 149  
 Filter property, 21  
 Font property, 41  
 FormBorderStyle property, 105-107  
 FormulaR1C1 property, 474  
 GridSize property, 120  
 Height property, 15  
 LayoutMode property, 120  
 MaximumSize property, 107  
 MaxLength property, 151  
 MinimumSize property, 107  
 Multiline property, 148  
 Name property, 11-13, 41  
 Opacity property, 135  
 PasswordChar property, 152  
 read-only properties, 60, 355-356

readable properties, creating, 354  
 referencing, 59-60  
 ScrollBars property, 150  
 ShowGrid property, 120-121  
 ShowInTaskbar property, 112  
 Size property, 42  
 SnapToGrid property, 120-121  
 StartPosition property, 109-111  
 TabIndex property, 133-134  
 TabStop property, 134  
 Text property, 13, 24  
 TextAlign property, 148  
 unchangeable properties, 60  
 viewing, 40  
 Width property, 15  
 WindowState property, 109-110  
 WordWrap property, 150  
 writable properties, creating, 354-355  
 write-only properties, creating, 355-356

#### **properties (projects), setting, 48-49**

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

#### **Publish Wizard, ClickOnce applications, 482-484, 488-489**

## **Q-R**

### **Quit button, 24-25**

### **radio buttons, 159-160**

### **Range objects, 474**

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

### **readable properties (objects), creating, 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, 430-433

data types, 429

deployments

debugging, 437

testing, 437

displaying options from, 434

hives, 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

keys

creating, 430-431

deleting, 431

getting values, 432-433

setting values, 432-433

## Registry (Windows)

saving options to, 435  
 stored options, using, 435-436  
 structure of, 428-429

### removing

list items  
   from enhanced lists, 186-187  
   from list boxes, 164-165  
 nodes from hierarchical lists, 190  
 project files, 49-50

### renaming files, 418

### Replace() function, 283

requirements (operational) for Visual Studio 2010, monitor resolution, 10

### reserved words, 246

### Resize events, 405

resolution (monitors), Visual Studio 2010 requirements, 10

### running programs, 25-27

### runtime errors, 326-327

## S

SaveFileDialog control, 409, 413-415

### saving

projects, 13  
 Registry options (Windows), 435

### SByte data type, 239

### scope

block (structure) scope, 254-255  
 defining, 254  
 global (namespace) scope, 256-257

limiting, 258  
 module-level scope, 255  
 name conflicts, 257  
 naming conventions, 260  
 of objects, browsing, 76  
 prefixes, 260  
 procedure-level (local) scope, 255  
 static scope, declaring variables, 258-259

### scrollable forms, 135-136

### scrollbars

adding to text boxes, 150  
 ScrollBars property, 150

### Select Case constructs, 299

Case Else statements, 303  
 Case statements, 300, 303  
 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, 481-482  
 Publish Wizard, 482-484, 488-489  
 testing, 486

### SetValue() method, 432

### shapes, drawing, 397

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, 120-121

### 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, 119-121  
 ShowGrid property, 120-121  
 SnapToGrid property, 120-121

### Size property, 42

### sizing

controls, 126  
   autosizing, 128-131  
 docked design windows, 35  
 forms, 15, 107-110  
 MaximumSize property, 107  
 MinimumSize property, 107

Snap to Lines, organizing controls on forms, 122

### SnapToGrid property, 120-121

### 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, 127

**SqlConnection** objects, 452-453  
**SqlDataAdapter** objects, creating, 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, 140-141  
**Startup** object, 140-141  
**static scope**, declaring variables, 258-259  
**static text**, displaying via Label control, 145-147  
**status bars**, Status Bar control, 213-214  
**Step** statements, 311  
**stopping programs**, writing code for, 24-25  
**storing Registry options** (Windows), using, 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**  
     finding, 53  
     further reading, 497  
**system colors**, 393-396  
**system date/time**, retrieving, 288  
**System** namespace, 494  
**System.Data** namespace, 452

## T

**Tab** control, 177-180  
**tab orders**, creating for controls, 132-134  
**tabbed dialog boxes**, creating, 177-180  
**TabIndex** property, 133-134  
**TabStop** property, 134  
**taskbar**  
     forms, preventing from appearing, 112  
     ShowInTaskbar property, 112  
**technical support**  
     finding, 53  
     further reading, 497  
**terminating programs**, writing code for, 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, 41  
     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, 283  
         retrieving text from left side of, 279-280  
         retrieving text from right side of, 280  
         retrieving text from within, 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 boxes**, 146  
     aligning text in, 148  
     common events, 153  
     limiting number of characters entered, 151  
     multiline text boxes, creating, 148-149

**text boxes**

- 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, **239-240**, 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), changing text in, 13**
  - toolbars, 37**
    - displaying, 37
    - hiding, 37
    - Toolbar control, 211
    - ToolStrip control, 208-210
  - toolbox controls, adding to forms, 38-39**
    - double-clicking controls in toolbox, 118
    - dragging controls from toolbox, 118
  - Toolbox window (Visual Studio 2010), opening/closing, 10**
  - transparent forms, 135**
  - triggering (invoking)**
    - events, 80
    - objects, 81
    - OS (operating systems), 82
    - user interaction, 81
    - methods (objects), 66
  - True statements, 274**
  - Try...Catch...Finally structures, creating error handlers, 336-338**
  - Try...End Try structures, 337-338**
    - Exit Try statements, 340
  - two button controls, 17**
  - type (object) libraries, 470**
- U**
- UInteger data type, 239**
  - ULong data type, 239**
  - uninstalling applications, 486**
  - unloading forms, 112-113**
  - user controls, defining, 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, 369-370
  - mouse events, 382-385
  - user information, obtaining, 377-379
  - user interfaces, creating (event building example), 87-88**
  - 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 array variables, 251-252
- storing values in, 51
- strict typing, 247-250
- viewing
  - descriptions, 44
  - properties (objects), 40
- visible controls, adding to forms, 17-18
- Visual Basic**
  - customizing, 32
    - docking design windows, 34-35
    - floating design windows, 34
    - hiding/displaying design windows, 33, 36
  - navigating, 32
  - relationship to .NET Framework, 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, opening/closing, 10

## W

- While...End loops, 320**
- Width property, 15**
- Window system color, 395**
- windows (Visual Studio 2010), 10**
- Windows Forms Applications, creating, 8-9**
- Windows Registry, 427**
  - accessing via
    - My.Computer.Registry object, 430-433
  - data types, 429
  - deployments
    - debugging, 437
    - testing, 437
  - displaying options from, 434
  - hives, 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
  - keys
    - creating, 430-431
    - deleting, 431
    - getting values, 432-433
    - setting values, 432-433
  - saving options to, 435
  - stored options, using, 435-436
  - structure of, 428-429

- WindowState property, 109-110**
- Word (MS) automation, 476**
  - instances of automation server creation, 477
  - library reference creation, 476-477
- WordWrap property, 150**
- workbooks (Excel)**
  - cell manipulation in, 473-475
  - creating, 473
- writable properties (objects), creating, 354-355**
- Write() method, 440**
- write-only properties (objects), creating, 355-356**
- WriteLine() method, 440-441**
- writing**
  - code, 21
    - browsing files example, 22-24
    - terminating programs example, 24-25
  - text files, 439-441

## X-Y-Z

- Xor operator, 276-277**

- yes/no options via check boxes, 156**