

James Foxall

Sams **Teach Yourself**

# Visual Basic® 2015

in **24**  
**Hours**

**SAMS**

FREE SAMPLE CHAPTER



SHARE WITH OTHERS

James Foxall

Sams **Teach Yourself**  
**Visual Basic®**  
**2015**

in **24**  
**Hours**

**SAMS**

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

## **Visual Basic® 2015 in 24 Hours, Sams Teach Yourself**

Copyright © 2016 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: 978-0-672-33745-1

ISBN-10: 0-672-33745-2

Library of Congress Control Number: 2015907842

Printed in the United States of America

First Printing August 2015

### **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 CD or programs accompanying it.

### **Special Sales**

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at [corpsales@pearsoned.com](mailto:corpsales@pearsoned.com) or (800) 382-3419.

For government sales inquiries, please contact [governmentsales@pearsoned.com](mailto:governmentsales@pearsoned.com).

For questions about sales outside the U.S., please contact [international@pearsoned.com](mailto:international@pearsoned.com).

### **Editor-in-Chief**

Greg Wiegand

### **Acquisitions Editor**

Joan Murray

### **Development Editor**

Mark Renfrow

### **Managing Editor**

Sandra Schroeder

### **Project Editor**

Mandie Frank

### **Copy Editor**

Keith Cline

### **Indexer**

Lisa Stumpf

### **Proofreader**

Katie Matejka

### **Technical Editor**

Lucian Wischik

### **Editorial Assistant**

Cindy Teeters

### **Designer**

Mark Shirar

### **Senior Compositor**

Gloria Schurick

# Contents at a Glance

Introduction .....	xvii
--------------------	------

## **Part I: The Visual Basic 2015 Environment**

<b>HOURL 1</b> Jumping in with Both Feet: A Visual Basic 2015 Programming Tour ....	1
<b>2</b> Navigating Visual Basic 2015 .....	31
<b>3</b> Understanding Objects and Collections .....	63
<b>4</b> Understanding Events .....	87

## **Part II: Building a User Interface**

<b>HOURL 5</b> Building Forms: The Basics .....	107
<b>6</b> Building Forms: Advanced Techniques .....	131
<b>7</b> Working with Traditional Controls .....	163
<b>8</b> Using Advanced Controls .....	193
<b>9</b> Adding Menus and Toolbars to Forms .....	215

## **Part III: Making Things Happen—Programming**

<b>HOURL 10</b> Creating and Calling Code Procedures .....	239
<b>11</b> Using Constants, Data Types, Variables, and Arrays .....	259
<b>12</b> Performing Arithmetic, String Manipulation, and Date/Time Adjustments .....	291
<b>13</b> Making Decisions in Visual Basic Code .....	313
<b>14</b> Looping for Efficiency .....	329
<b>15</b> Debugging Your Code .....	343
<b>16</b> Designing Objects Using Classes .....	371
<b>17</b> Interacting with Users .....	391
<b>18</b> Working with Graphics .....	413

**Part IV: Working with Data**

<b>HOURL 19</b>	Performing File Operations .....	435
<b>20</b>	Working with the Registry and Text Files .....	457
<b>21</b>	Working with a Database .....	483
<b>22</b>	Printing .....	505
<b>23</b>	Sending Emails .....	529

**Part V: Deploying Solutions and Beyond**

<b>HOURL 24</b>	Deploying Applications .....	545
<b>APPENDIX A</b>	The 10,000-Foot View .....	559
	Index .....	567

# Table of Contents

Introduction	xvii
--------------	------

## Part I: The Visual Basic 2015 Environment

<b>Hour 1: Jumping in with Both Feet: A Visual Basic 2015 Programming Tour</b>	<b>1</b>
Starting Visual Basic 2015	2
Creating a New Project	3
Understanding the Visual Studio 2015 Environment	6
Changing the Characteristics of Objects	7
Adding Controls to a Form	13
Designing an Interface	15
Writing the Code Behind an Interface	20
Running a Project	24
Summary	27
Q&A	28
Workshop	28
Exercises	29
<b>Hour 2: Navigating Visual Basic 2015</b>	<b>31</b>
Using the Visual Basic 2015 Start Page	31
Navigating and Customizing the Visual Basic Environment	34
Working with Toolbars	40
Adding Controls to a Form Using the Toolbox	41
Setting Object Properties Using the Properties Window	43
Managing Projects	50
A Quick-and-Dirty Programming Primer	57
Getting Help	59
Summary	60
Q&A	60
Workshop	61
Exercises	61

<b>Hour 3: Understanding Objects and Collections</b>	<b>63</b>
Understanding Objects	64
Understanding Properties	64
Understanding Methods	72
Building a Simple Object Example Project	73
Understanding Collections	79
Using the Object Browser	82
Summary	84
Q&A	84
Workshop	85
Exercises	85
<b>Hour 4: Understanding Events</b>	<b>87</b>
Understanding Event-Driven Programming	87
Building an Event Example Project	97
Keeping Event Names Current	103
Summary	103
Q&A	104
Workshop	104
Exercises	105
<b>Part II: Building a User Interface</b>	
<b>Hour 5: Building Forms: The Basics</b>	<b>107</b>
Changing a Form's Name	108
Changing a Form's Appearance	109
Showing and Hiding Forms	122
Summary	128
Q&A	128
Workshop	129
Exercises	130
<b>Hour 6: Building Forms: Advanced Techniques</b>	<b>131</b>
Working with Controls	131
Creating Topmost Nonmodal Windows	151
Creating Transparent Forms	151

Creating Scrollable Forms . . . . .	152
Creating MDI Forms . . . . .	154
Setting the Startup Form . . . . .	158
Summary . . . . .	159
Q&A . . . . .	160
Workshop . . . . .	160
Exercises . . . . .	161
<b>Hour 7: Working with Traditional Controls</b>	<b>163</b>
Displaying Static Text with the Label Control . . . . .	163
Allowing Users to Enter Text Using a Text Box . . . . .	164
Creating Buttons . . . . .	172
Creating Containers and Groups of Option Buttons . . . . .	176
Displaying a List with the List Box . . . . .	180
Creating Drop-Down Lists Using the Combo Box . . . . .	188
Summary . . . . .	190
Q&A . . . . .	191
Workshop . . . . .	191
Exercises . . . . .	192
<b>Hour 8: Using Advanced Controls</b>	<b>193</b>
Creating Timers . . . . .	193
Creating Tabbed Dialog Boxes . . . . .	197
Storing Pictures in an Image List Control . . . . .	200
Building Enhanced Lists Using the List View Control . . . . .	202
Creating Hierarchical Lists Using the Tree View Control . . . . .	207
Summary . . . . .	211
Q&A . . . . .	212
Workshop . . . . .	212
Exercises . . . . .	213
<b>Hour 9: Adding Menus and Toolbars to Forms</b>	<b>215</b>
Building Menus . . . . .	215
Using the Toolbar Control . . . . .	229
Creating a Status Bar . . . . .	235
Summary . . . . .	237



Q&A .....	237
Workshop .....	238
Exercises .....	238

### **Part III: Making Things Happen—Programming**

<b>Hour 10: Creating and Calling Code Procedures</b> .....	<b>239</b>
Creating Visual Basic Code Modules .....	239
Writing Code Procedures .....	242
Calling Code Procedures .....	248
Exiting Procedures .....	254
Avoiding Infinite Recursion .....	255
Summary .....	256
Q&A .....	257
Workshop .....	257
Exercises .....	258
<b>Hour 11: Using Constants, Data Types, Variables, and Arrays</b> .....	<b>259</b>
Understanding Data Types .....	260
Defining and Using Constants .....	263
Declaring and Referencing Variables .....	266
Working with Arrays .....	273
Determining Scope .....	276
Declaring Variables of Static Scope .....	281
Using Variables in Your Picture Viewer Project .....	282
Renaming Variables .....	286
Summary .....	287
Q&A .....	288
Workshop .....	288
Exercises .....	289
<b>Hour 12: Performing Arithmetic, String Manipulation, and Date/Time Adjustments</b> .....	<b>291</b>
Performing Basic Arithmetic Operations with Visual Basic .....	291
Comparing Equalities .....	295
Understanding Boolean Logic .....	296

Manipulating Strings . . . . .	298
Working with Dates and Times . . . . .	304
Summary . . . . .	309
Q&A . . . . .	310
Workshop . . . . .	310
Exercises . . . . .	311
<b>Hour 13: Making Decisions in Visual Basic Code</b>	<b>313</b>
Making Decisions Using <code>If . . . Then</code> . . . . .	313
Branching Within a Procedure Using <code>GoTo</code> . . . . .	324
Summary . . . . .	326
Q&A . . . . .	327
Workshop . . . . .	327
Exercises . . . . .	328
<b>Hour 14: Looping for Efficiency</b>	<b>329</b>
Looping a Specific Number of Times Using <code>For . . . Next</code> . . . . .	329
Using <code>Do . . . Loop</code> to Loop an Indeterminate Number of Times . . . . .	336
Summary . . . . .	341
Q&A . . . . .	341
Workshop . . . . .	342
Exercises . . . . .	342
<b>Hour 15: Debugging Your Code</b>	<b>343</b>
Adding Comments to Your Code . . . . .	344
Identifying the Two Basic Types of Errors . . . . .	346
Using Visual Basic's Debugging Tools . . . . .	349
Breaking Only When a Condition Is Met . . . . .	358
Breaking Only When a Breakpoint Is Hit a Certain Number of Times . . . . .	359
Sending Messages to the Output Window Using Tracepoints . . . . .	360
Writing an Error Handler Using <code>Try . . . Catch . . . Finally</code> . . . . .	360
Summary . . . . .	368
Q&A . . . . .	368
Workshop . . . . .	368
Exercises . . . . .	369

<b>Hour 16: Designing Objects Using Classes</b>	<b>371</b>
Understanding Classes	372
Instantiating Objects from Classes	381
Summary	388
Q&A	388
Workshop	388
Exercises	389
<b>Hour 17: Interacting with Users</b>	<b>391</b>
Displaying Messages Using the <code>MessageBox.Show()</code> Function	391
Creating Custom Dialog Boxes	398
Using <code>InputDialog()</code> to Get Information from a User	401
Interacting with the Keyboard	404
Using the Common Mouse Events	406
Summary	409
Q&A	410
Workshop	410
Exercises	411
<b>Hour 18: Working with Graphics</b>	<b>413</b>
Understanding the <code>Graphics</code> Object	413
Working with Pens	416
Using System Colors	417
Working with Rectangles	421
Drawing Shapes	422
Drawing Text	423
Persisting Graphics on a Form	425
Building a Graphics Project Example	425
Summary	432
Q&A	432
Workshop	432
Exercises	433

**Part IV: Working with Data**

<b>Hour 19: Performing File Operations</b>	<b>435</b>
Using the OpenFileDialog and SaveFileDialog Controls	435
Manipulating Files with the File Object	443
Manipulating Directories with the Directory Object	452
Summary	453
Q&A	454
Workshop	454
Exercises	455
<b>Hour 20: Working with the Registry and Text Files</b>	<b>457</b>
Working with the Registry	457
Reading and Writing Text Files	470
Summary	480
Q&A	481
Workshop	481
Exercises	482
<b>Hour 21: Working with a Database</b>	<b>483</b>
Introducing ADO.NET	484
Manipulating Data	491
Summary	502
Q&A	502
Workshop	503
Exercises	503
<b>Hour 22: Printing</b>	<b>505</b>
Preparing the Picture Viewer Project	506
Printing and Previewing a Document	509
Changing Printer and Page Settings	519
Scaling Images to Fit a Page	522
Summary	527
Q&A	528
Workshop	528
Exercises	528

<b>Hour 23: Sending Emails</b>	<b>529</b>
Understanding the Classes Used to Send Emails . . . . .	530
Sending Email from Your Picture Viewer Application . . . . .	530
Summary . . . . .	543
Q&A . . . . .	544
Workshop . . . . .	544
Exercises . . . . .	544
<b>Part V: Deploying Solutions and Beyond</b>	
<b>Hour 24: Deploying Applications</b>	<b>545</b>
Understanding ClickOnce Technology . . . . .	545
Using the Publish Wizard to Create a ClickOnce Application . . . . .	547
Testing Your Picture Viewer ClickOnce Install Program . . . . .	552
Uninstalling an Application You've Distributed . . . . .	553
Setting Advanced Options for Creating ClickOnce Programs . . . . .	556
Summary . . . . .	557
Q&A . . . . .	557
Workshop . . . . .	557
Exercises . . . . .	558
<b>Appendix A: The 10,000-Foot View</b>	<b>559</b>
The .NET Framework . . . . .	559
Common Language Runtime . . . . .	560
Microsoft Intermediate Language . . . . .	560
Namespaces . . . . .	562
Common Type System . . . . .	563
Garbage Collection . . . . .	564
Further Reading . . . . .	564
Summary . . . . .	565
<b>Index</b>	<b>567</b>

# About the Author

**James Foxall** is President & CEO of Tigerpaw Software, a commercial software company providing complete business automation to more than 40,000 users in 28 countries in the IT/Networking, Telecommunications, Systems Integrator, Security, and Point of Sale industries. In his current role as President and CEO, James provides the vision and management to keep Tigerpaw focused on its customers and properly serving its markets.

James has a Masters degree in Business Administration and a BS degree in Management of Information Systems. Devoted to creating better businesses through technology, James has written 15 books, which have been published in over a dozen languages around the world. He is considered an authority on business process improvement and application interface and behavior standards of Windows applications, and serves the business community as an international speaker on automating business processes in the SMB environment.

# Dedication

*This book is dedicated to Connie Derry,  
for giving me room to “express myself” in my writing.*

# Acknowledgments

I would like to thank my kids Tess and Ethan, for reminding me there is more to life than work, and for giving me something to work for.

I would also like to thank all the great people at Sams for their faith, 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.

We welcome your comments. You can email or write to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

*Please note that we cannot help you with technical problems related to the topic of this book.*

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

Email: [consumer@sampublishing.com](mailto:consumer@sampublishing.com)

Mail: Sams Publishing  
ATTN: Reader Feedback  
800 East 96th Street  
Indianapolis, IN 46240 USA

## Reader Services

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



*This page intentionally left blank*

# Introduction

Visual Basic 2015 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.

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 2015 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 2015 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,” teaches you how to add emailing capabilities to your projects, and shows you how to distribute an application that you’ve created to an end user’s computer. In Appendix A, “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 allows 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 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? boxes highlight information that can make your Visual Basic programming more effective.

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 1

## Jumping in with Both Feet: A Visual Basic 2015 Programming Tour

---

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

- ▶ Building a simple (yet functional) Visual Basic application
- ▶ Letting a user browse a hard drive
- ▶ Displaying a picture from a file on disk
- ▶ Getting familiar with some programming lingo
- ▶ Learning about the Visual Studio 2015 IDE

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

Producing large, commercial solutions is accomplished by way of a series of small steps. After you've finished this hour, you'll have a feel for the overall development process and will have taken the first step toward becoming an accomplished programmer. In fact, you will build on the examples in this hour in subsequent hours. By the time you complete this book, you will have built a robust application, complete with resizable screens, an intuitive interface including menus and toolbars, manipulation of the Windows Registry, and robust code with professional error handling. But I'm getting ahead of myself.

In this hour, you complete a quick tour of Visual Basic that takes you step by step through creating a complete, albeit small, Visual Basic program. Most introductory programming books start by having the reader create a simple Hello World program. I've yet to see a Hello World program that's the least bit helpful. (They usually do nothing more than print `hello world` to the screen—what fun!) So, instead, you create a Picture Viewer application that lets you view pictures on your computer. You learn how to let a user browse for a file and how to display a selected picture file on the screen. The techniques you learn in this hour will come in handy in

many real-world applications that you'll create, but the goal of this hour is for you to realize just how much fun it is to program using Visual Basic 2015.

## Starting Visual Basic 2015

Before you begin creating programs in Visual Basic 2015, you should be familiar with the following terms:

- ▶ **Distributable component:** The final, compiled version of a project. Components can be distributed to other people and other computers, and they don't require the Visual Basic 2015 development environment (the tools you use to create a .NET program) to run (although they do require the .NET runtime, as discussed in Hour 23, "Deploying Applications"). Distributable components are often called *programs*. In Hour 23, you learn how to distribute the Picture Viewer program that you're about to build to other computers.
- ▶ **Project:** A collection of files that can be compiled to create a distributable component (program). There are many types of projects, and complex applications might consist of multiple projects, such as Windows application projects, and support dynamic link library (DLL) projects.
- ▶ **Solution:** A collection of projects and files that make up an application or component.

### BY THE WAY

---

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

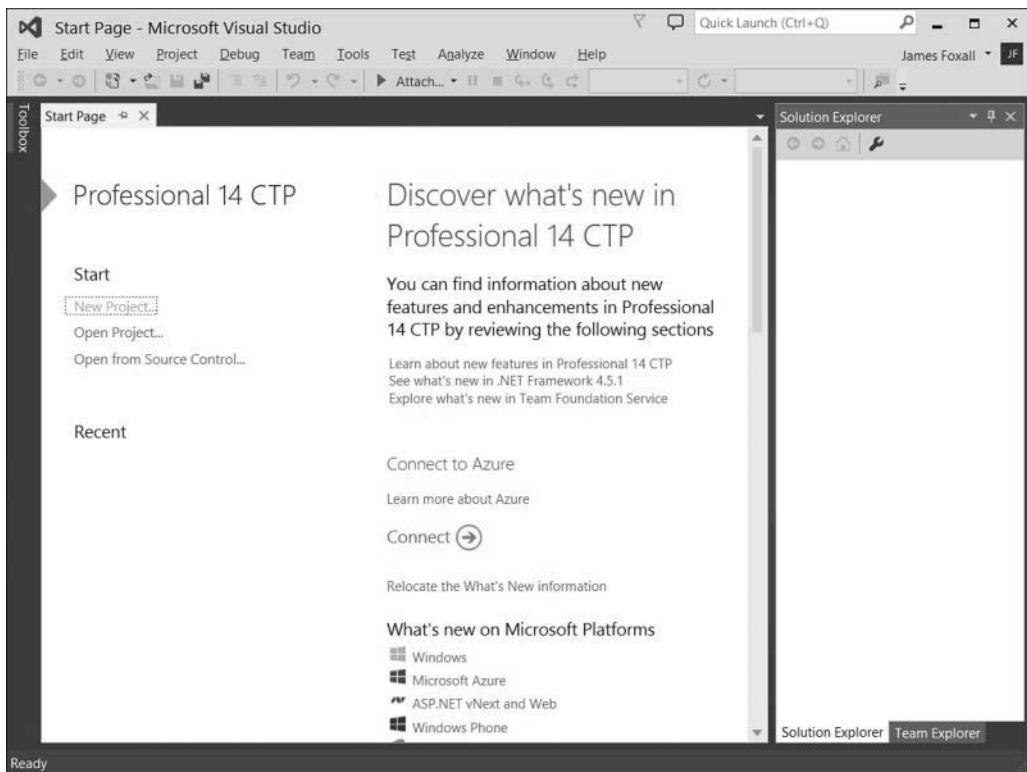
---

Visual Studio 2015 is a complete development environment, and it's called the *IDE* (short for *integrated development environment*). The IDE is the design framework in which you build applications; every tool you need to create your Visual Basic projects is accessed from within the Visual Basic IDE. Again, Visual Studio 2015 supports development using many different languages, Visual Basic being the most popular. The environment itself is not Visual Basic, but the language you use within Visual Studio 2015 *is* Visual Basic. To work with Visual Basic projects, you first start the Visual Studio 2015 IDE.

Start Visual Studio 2015 now by choosing Microsoft Visual Basic 2015 Express Edition from the Start/Programs menu. If you are running the full retail version of Visual Studio, your shortcut may have a different name. In this case, locate the shortcut on the Start menu and click it once to start the Visual Studio 2015 IDE.

## Creating a New Project

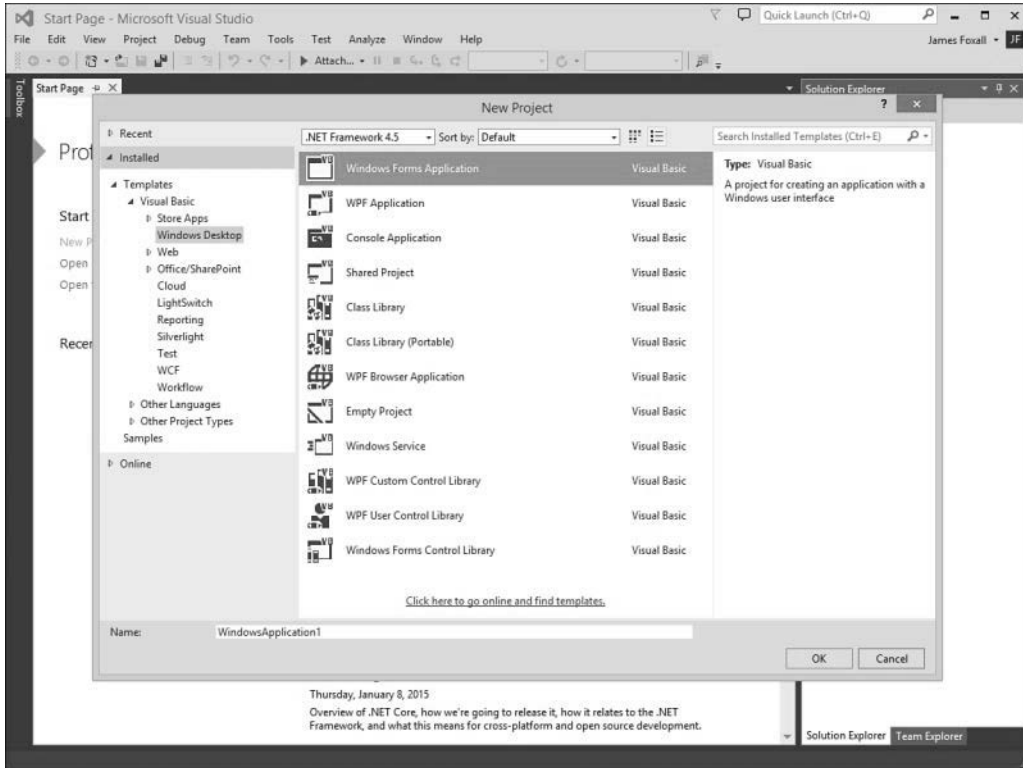
When you first start Visual Studio 2015, you see the Start Page tab within the IDE, as shown in Figure 1.1. You can open projects created previously or create new projects from this Start page. For this quick tour, you'll create a new Windows application, so select File, New Project to display the New Project dialog box shown in Figure 1.2.



**FIGURE 1.1**

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





**FIGURE 1.2**  
The New Project dialog box enables you to create many types of .NET projects.

## BY THE WAY

Your Start page might look a little different than the one shown in Figure 1.1—depending on what version of Visual Studio you are using.

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

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

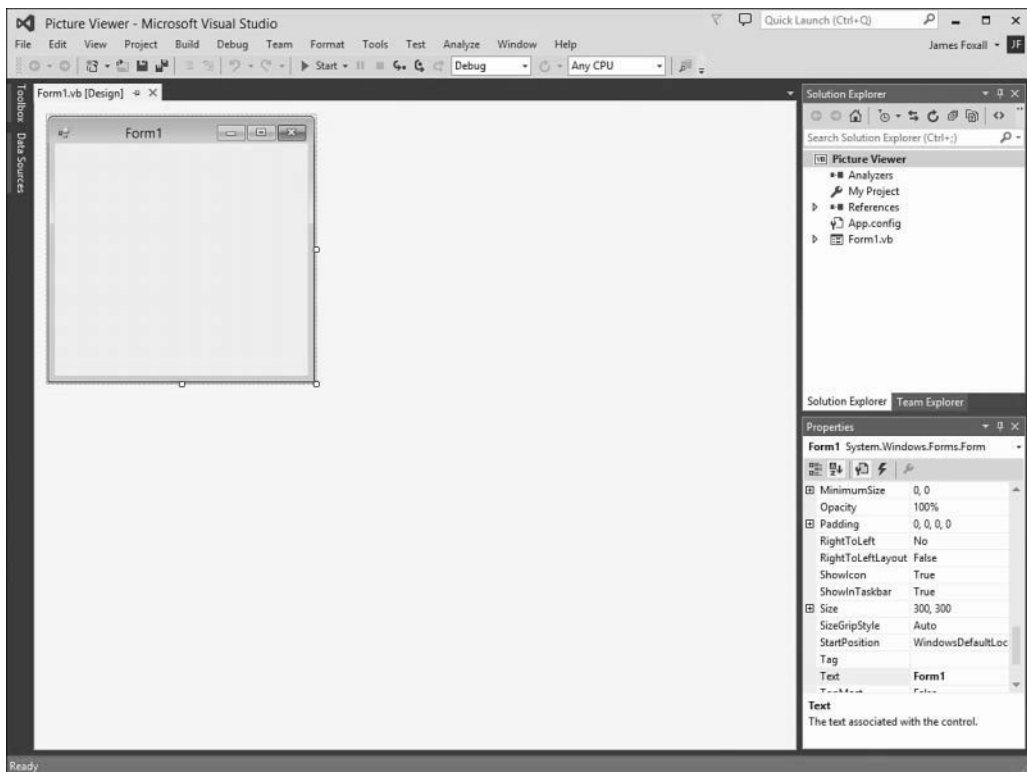
1. Click Windows Desktop in the tree on the left.
2. Click the Windows Forms Application item to select it.

3. At the bottom of the New Project dialog box is a Name text box. This is where, oddly enough, you specify the name of the project you're creating. Enter **Picture Viewer** in the Name text box.
4. Click OK to create the project.

## DID YOU KNOW?

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

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



**FIGURE 1.3**

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

---

## BY THE WAY

---

Within Visual Studio 2015, *form* is the term given to the design-time view of a window that can be displayed to a user.

---

Your Visual Studio 2015 environment might look different from that shown in the figures in this hour, depending on the edition of Visual Studio 2015 you're using, whether you've already played with Visual Studio 2015, and other factors, such as your monitor's resolution. However, all the elements discussed in this hour exist in all editions of Visual Studio 2015 that support desktop development (as opposed to store applications). If a window shown in a figure doesn't appear in your IDE, use the View menu to display it.

---

## BY THE WAY

---

To create a program that can be run on another computer, you start by creating a project and then compiling the project into a component such as an *executable* (a program a user can run) or a *DLL* (a component that can be used by other programs and components). The compilation process is discussed in detail in Hour 24, "Deploying Applications." The important thing to note at this time is that when you hear someone refer to *creating or writing a program*, just as you're creating the Picture Viewer program now, that person is referring to the completion of all steps up to and including compiling the project to a distributable file.

---

# Understanding the Visual Studio 2015 Environment

The first time you run Visual Studio 2015, you'll notice that the IDE contains a number of windows, such as the Properties window on the lower-right, which is used to view and set properties of objects. In addition to these windows, the IDE contains a number of tabs, such as the vertical Toolbox and Data Source tabs on the left edge of the IDE (refer to Figure 1.3). Try this now: Click the Toolbox tab to display the Toolbox window (clicking a tab displays an associated window). To hide the window, click another window. To close the window (don't do this now), click the Close (X) button on the window's title bar.

You can adjust the size and position of any of these windows, and you can even hide and show them as needed. You'll learn how to customize your design environment in Hour 2, "Navigating Visual Basic 2015."

## WATCH OUT!

---

Unless specifically instructed to do so, don't double-click anything in the Visual Studio 2015 design environment. Double-clicking most objects produces an entirely different result than single-clicking does. If you mistakenly double-click an object on a form (discussed shortly), a code window appears. At the top of the code window is a set of tabs: one for the form design and one for the code. Click the tab for the form design to hide the code window and return to the form.

---

The Properties window on the right side of the design environment is perhaps the most important window in the IDE, and it's the one you'll use most often. If your computer display resolution is set to 1024×768, you can probably see only a few properties at this time. This makes it difficult to view and set properties as you create projects. All the screen shots in this book were captured on Windows 8 running at 1152×864 because of publishing constraints, but you should run at a higher resolution if you can. I highly recommend that you develop applications with Visual Basic at a screen resolution of 1280×768 or higher to have plenty of workspace. To change your display settings, right-click the desktop and select Screen Resolution. Keep in mind, however, that end users might be running at a lower resolution than you are using for development.

## Changing the Characteristics of Objects

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

## WATCH OUT!

---

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

---

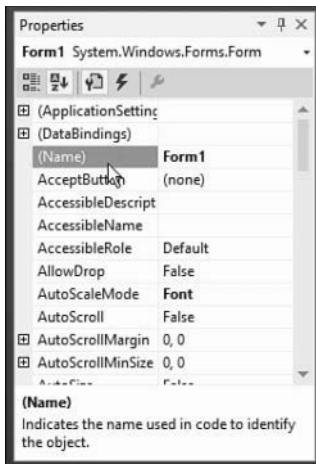
Every object has a distinct set of attributes known as *properties* (regardless of whether the object has a physical appearance). Properties define an object's characteristics. You have certain properties as a person, such as your height and hair color. Visual Basic objects have properties as well, such as `Height` and `BackColor`. When you create a new object, the first thing you need to

do is set its properties so that the object appears and behaves the way you want it to. To display an object's properties, click the object in its designer (the main work area in the IDE).

Click anywhere in the default form now (it's the window with the title Form1), and check to see that its properties are displayed in the Properties window. You'll know because the drop-down list box at the top of the Properties window contains the form's name: `Form1` is the object's name and `System.Windows.Forms.Form` is the object's type.

## Naming Objects

The property you should always set first when creating any new object is the `Name` property. Press F4 to display the Properties window (if it's not already visible), and scroll toward the top of the properties list until you see the `(Name)` property, as shown in Figure 1.4. If the `Name` property isn't one of the first properties listed, the Properties window is set to show properties categorically instead of alphabetically. You can show the list alphabetically by clicking the Alphabetical button that appears just above the properties grid.



**FIGURE 1.4**

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

---

### BY THE WAY

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

---

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

---

#### BY THE WAY

---

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

---

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

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

---

#### BY THE WAY

---

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

---

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

## Setting the Form's Text Property

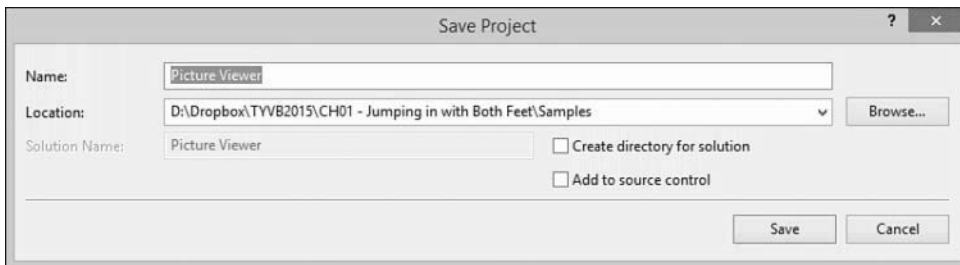
Notice that the text that appears in the form's title bar says Form1. Visual Basic sets the form's title bar to the name of the form *when it's first created*, but doesn't change it when you change the name of the form. The text on the title bar is determined by the value of the form's `Text` property.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

1. Click the form once more so that its properties appear in the Properties window.
2. Use the scrollbar in the Properties window to locate the `Text` property. If you're lucky, Visual Studio will have already selected this property for you.
3. Change the text to **Picture Viewer**. Press the Enter key or Tab key, or click a different property to commit your edit. You'll see the text on the form's title bar change.

## Saving a Project

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

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



**FIGURE 1.5**

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

## Giving the Form an Icon

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

---

### BY THE WAY

The following instructions assume that you have access to the source files for the examples in this book. They are available at <http://www.sampublishing.com>. You can also get these files in the Downloads section of my website at <http://www.jamesfoxall.com>. When you unzip the samples, a folder is created for each hour, and within each hour's folder are subfolders for the sample projects. You'll find the icon for this example in the folder Hour 01\Samples.

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

---

To give the form an icon, follow these steps:

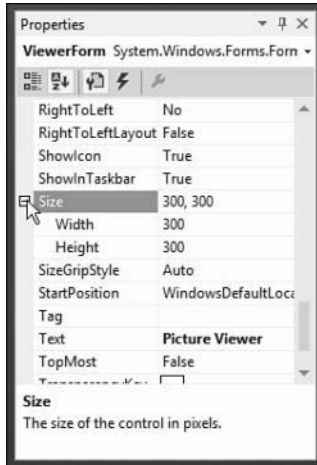
1. In the Properties window, click the `Icon` property to select it.
2. When you click the `Icon` property, a small button with three dots appears to the right of the property. Click this button.
3. Use the Open dialog box that appears to locate the `Picture Viewer.ico` file or another icon file of your choice. When you've found the icon, double-click it, or click it once to select it and then choose Open.

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

## Changing the Form's Size

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



**FIGURE 1.6**

Some properties can be expanded to show more specific properties.

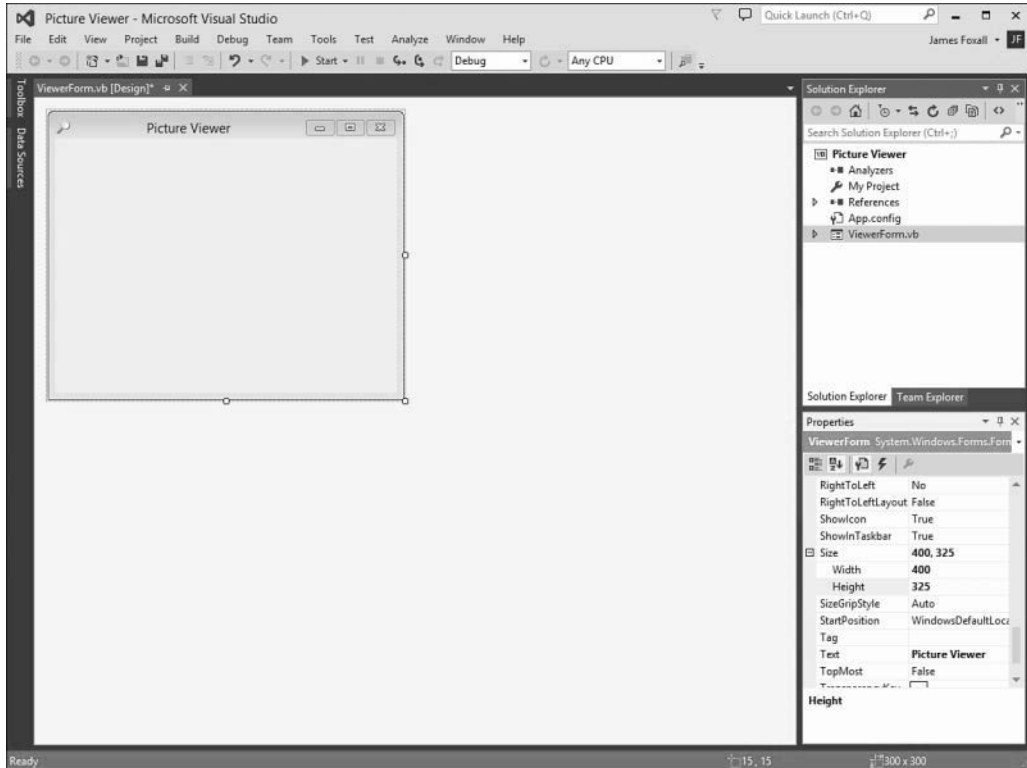
#### BY THE WAY

---

A pixel is a unit of measurement for computer displays; it's the smallest visible "dot" on the screen. The resolution of a display is always given in pixels, such as 800×600 or 1024×768. When you increase or decrease a property by 1 pixel, you're making the smallest possible visible change to the property.

---

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



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

## BY THE WAY

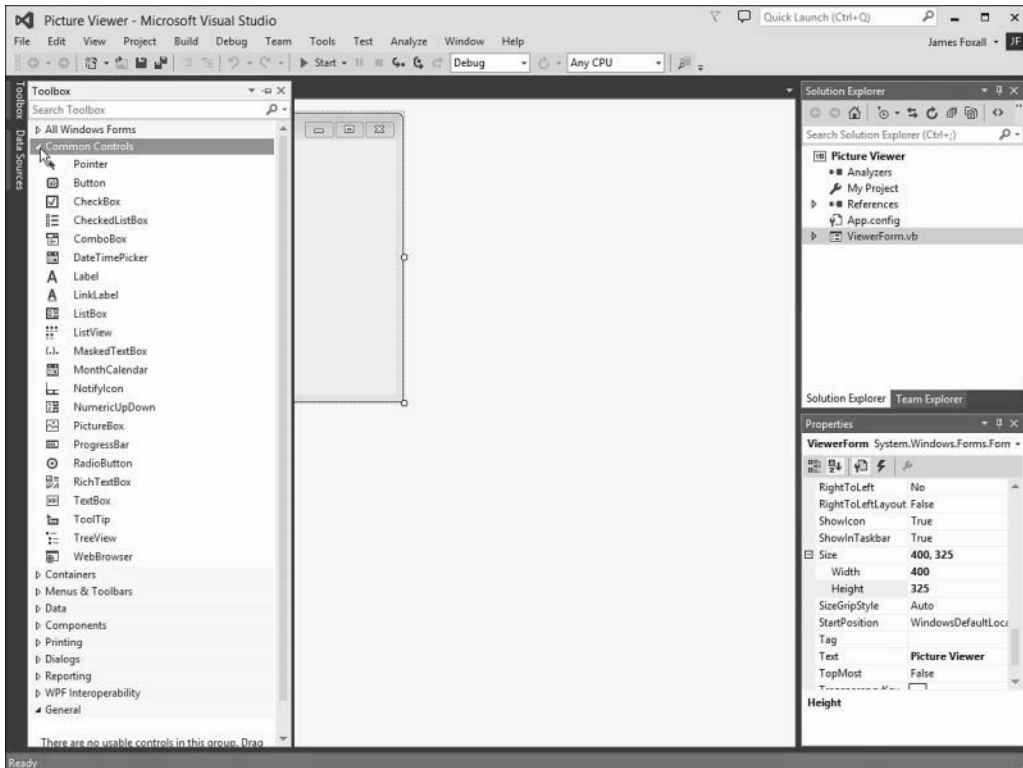
You can also resize a form by dragging its border, which you learn about in Hour 2, or by using code to change its properties, which you learn how to do in Hour 5.

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

## Adding Controls to a Form

Now that you've set the initial properties of your form, it's time to create a user interface by adding objects to the form. Objects that can be placed on a form are called *controls*. Some controls have a visible interface with which a user can interact, whereas others are always invisible to the user. You'll use controls of both types in this example. On the left side of the screen is a vertical

tab titled Toolbox. Click the Toolbox tab to display the Toolbox window to see the most commonly used controls, expanding the Common Controls section if necessary (see Figure 1.8). The toolbox contains all the controls available in the project, such as labels and text boxes.



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

Clicking off the toolbox makes it disappear. To make the toolbox stay visible, even when you click something else, click the little picture of a pushpin located in the toolbox's title bar.

I don't want you to add them yet (I'll walk you through the process), but your Picture Viewer interface will consist of the following controls:

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

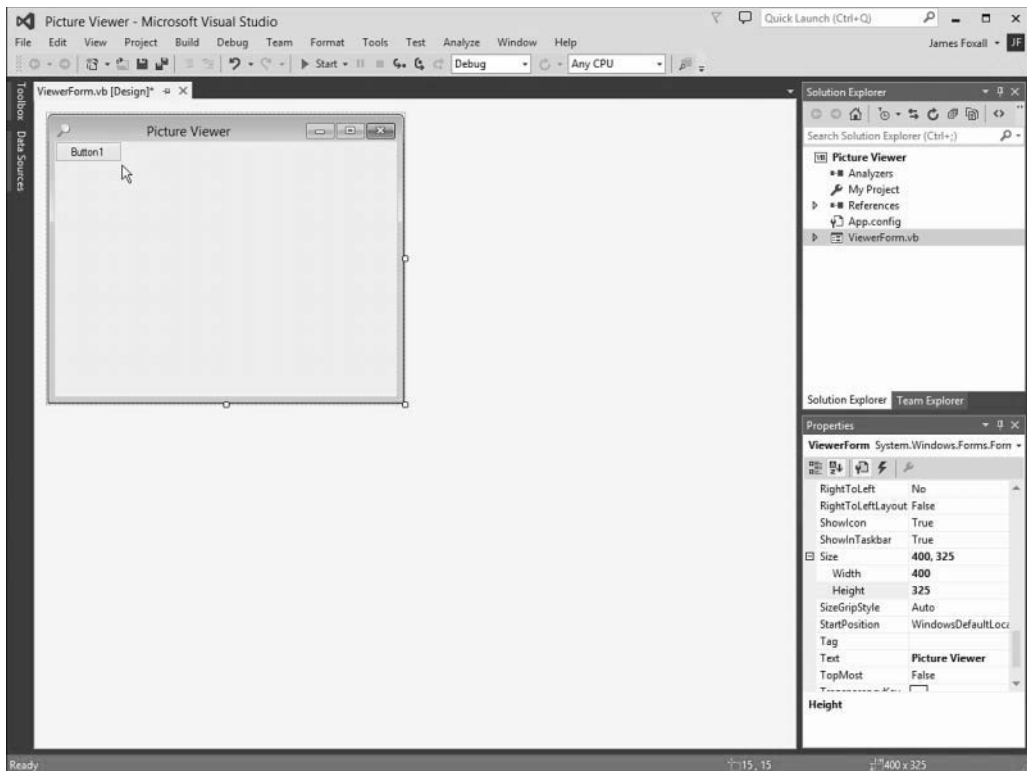
## Designing an Interface

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

### Adding a Visible Control to a Form

Start by adding a `Button` control to the form. Do this by double-clicking the `Button` item in the toolbox. Visual Basic creates a new button and places it in the upper-left corner of the form.

Click off the toolbox to make it go away so that you can see the new button control, as shown in Figure 1.9.



**FIGURE 1.9**

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

Using the Properties window, set the button's properties as shown in the following list. Because you clicked the form to make the toolbox go away, you need to click the button to select it and change its properties. Remember, when you view the properties alphabetically, the Name

property is listed first, so don't go looking for it down in the list; otherwise, you'll be looking a while.

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

#### BY THE WAY

If you see only the word Select on your button, chances are you've set your Windows fonts to a size larger than standard. Right-click the desktop and choose Personalize from the shortcut menu that appears. Next, click Display in the lower-left corner and change the font size on the Display dialog box that appears.

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

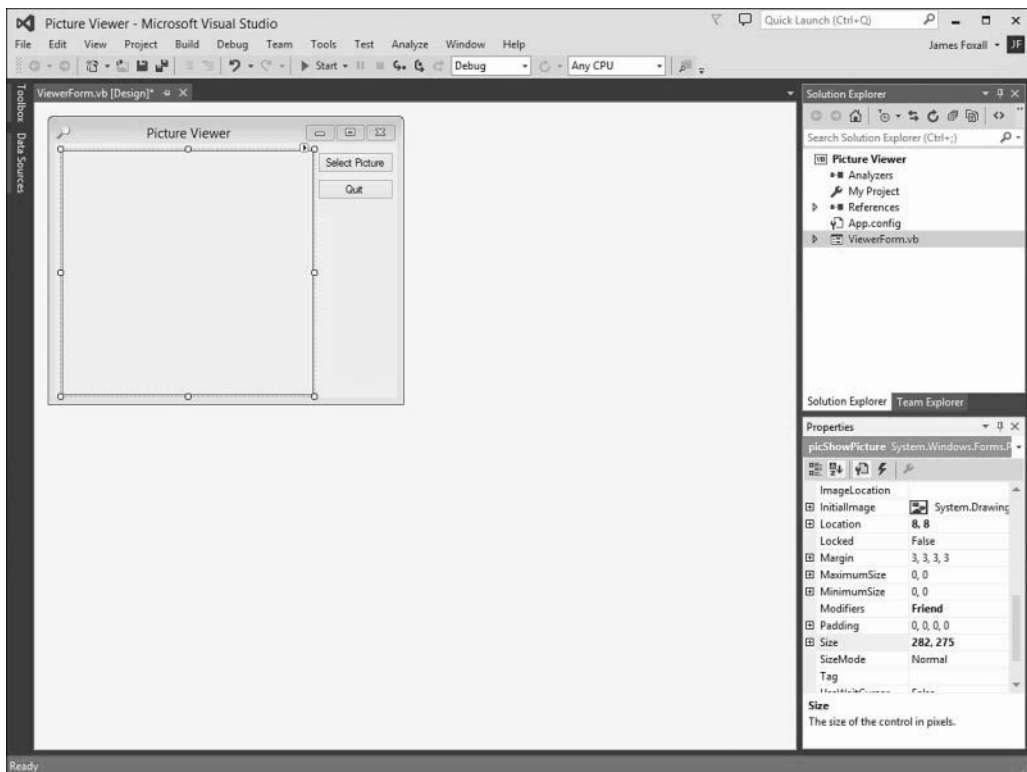
To do this, right-click the Select Picture button, and choose Copy from its context menu. Next, right-click anywhere on the form, and choose Paste from the form's shortcut menu. (You can also use the keyboard shortcuts Ctrl+C to copy and Ctrl+V to paste.) The new button appears centered on the form, and it's selected by default. Notice that it retains almost all the properties of the original button, but the name has been reset. Change the properties of the new button as follows:

<b>Property</b>	<b>Value</b>
Name	btnQuit
Location	295,40
Text	Quit

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

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

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



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

## Adding an Invisible Control to a Form

All the controls you've used so far sit on a form and have a physical appearance when a user runs the application. Not all controls have a physical appearance, however. Such controls, called *nonvisual controls* (or *invisible-at-runtime controls*), aren't designed for direct user interactivity. Instead, they're designed to give you, the programmer, functionality beyond the standard features of Visual Basic.

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

---

### BY THE WAY

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

---

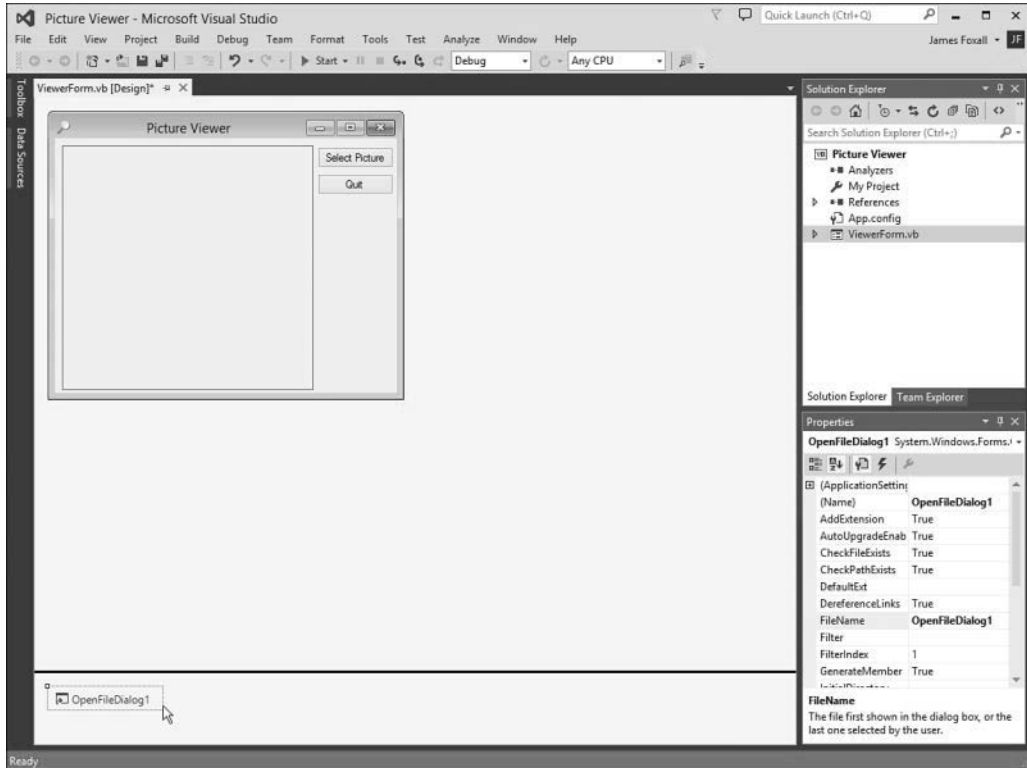
Display the toolbox and scroll down until you can see the Dialogs category. Expand the Dialogs category and then double-click the `OpenFileDialog` to add it to your form. Note that the control isn't placed on the form; rather, it appears in a special area below the form (see Figure 1.11).

This happens because the `OpenFileDialog` control has no form interface to display to the user. It does have an interface (a dialog box) that you can display as necessary, but it has nothing to display directly on a form.

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

Property	Value
Name	<code>ofdSelectPicture</code>
Filename	<code>&lt;make empty&gt;</code>
Filter	<code>PNG Files *.png Windows Bitmaps *.bmp JPEG Files *.jpg</code>
Title	<code>Select Picture</code>

---



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

## BY THE WAY

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

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

The graphical interface for your Picture Viewer program is now finished. If you pinned the toolbox open, click the pushpin on the title bar of the toolbox now to close it. Click Save All on the toolbar now to save your work.



## Writing the Code Behind an Interface

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

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

### Letting a User Browse for a File

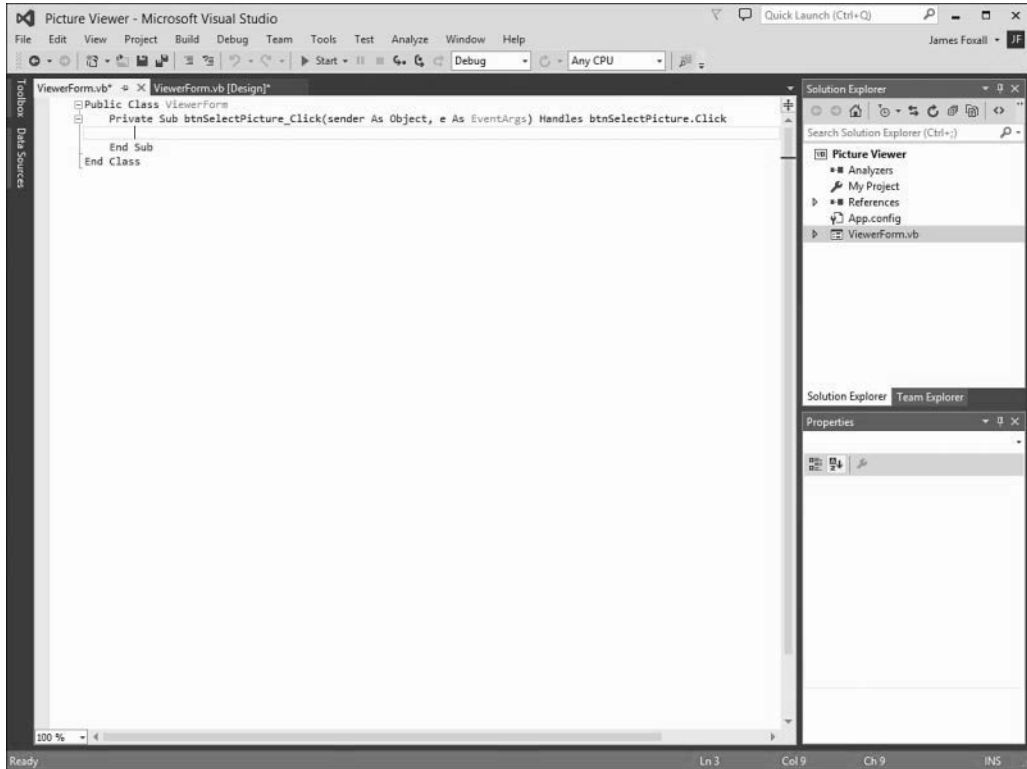
The first bit of code you’ll write enables users to browse their hard drives, select a picture file, and then see the selected picture in the `PictureBox` control. This code executes when the user clicks the Select Picture button; therefore, it’s added to the `Click` event of that button.

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

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

It’s important that you get in the habit of commenting your code, so the first statement you’ll enter is a comment. Beginning a statement with an apostrophe (‘) designates that statement as a comment. The compiler won’t do anything with the statement, so you can enter whatever text you want after the apostrophe. Type the following statement exactly as it appears, and press the Enter key at the end of the line:

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

**FIGURE 1.12**

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

The next statement you enter triggers a method of the `OpenFileDialog` control that you added to the form. Think of a method as a mechanism to make a control do something. The `ShowDialog()` method tells the control to show its Open dialog box and let the user select a file. The `ShowDialog()` method returns a value that indicates its success or failure, which you'll then compare to a predefined result (`DialogResult.OK`). Don't worry too much about what's happening here; you'll learn the details of all this in later hours. The sole purpose of this hour is to get your feet wet. In a nutshell, the `ShowDialog()` method is invoked to let a user browse for a file. If the user selects a file, more code is executed. Of course, there's a lot more to using the `OpenFileDialog` control than presented in this basic example, but this simple statement gets the job done. Enter the following statement and press Enter to commit the code. (Don't worry about capitalization; Visual Basic will fix the case for you.)

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

## BY THE WAY

---

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

---

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

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

## DID YOU KNOW?

---

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

---

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

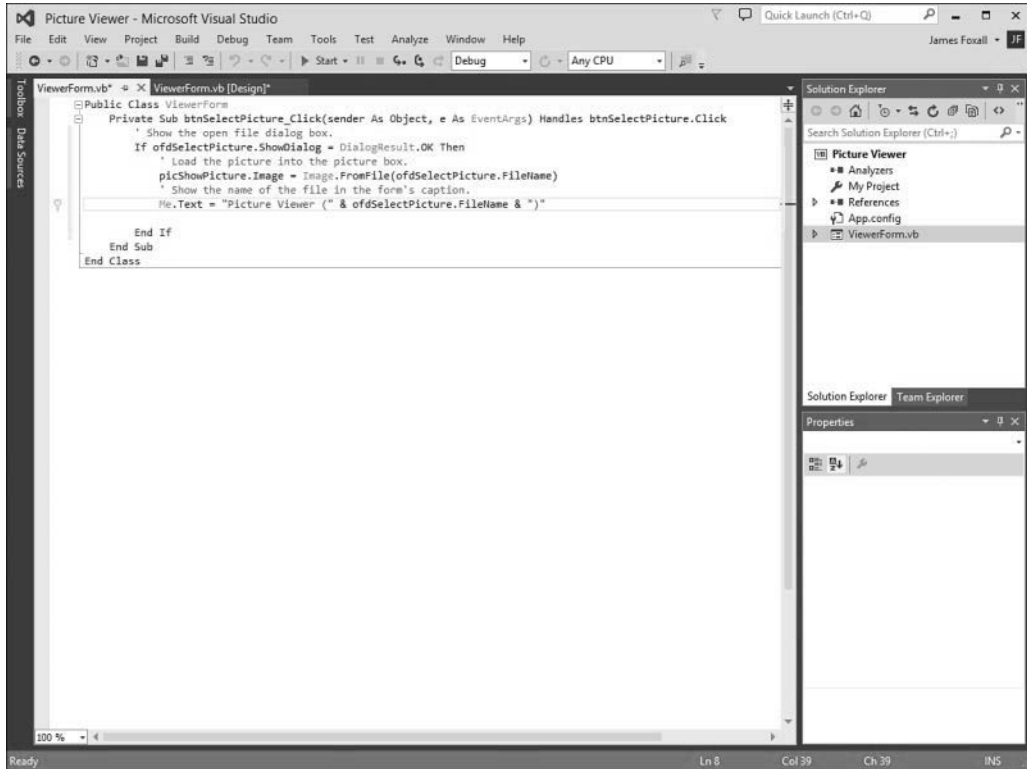
Type the following statement and press Enter:

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

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

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

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

**FIGURE 1.13**

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

## Terminating a Program Using Code

The last bit of code you'll write terminates the application when the user clicks the Quit button. To do this, you need to access the `Click` event handler of the `btnQuit` button. At the top of the code window are two tabs. The current tab says `ViewerForm.vb*`. This tab contains the code window for the form that has the filename `ViewerForm.vb`. Next to this is a tab that says `ViewerForm.vb [Design]*`. Click this tab to switch from Code view to the form designer. If you receive an error when you click the tab, the code you entered contains an error, and you need to edit it to make it the same, as shown in Figure 1.13. After the form designer appears, double-click the Quit button to access its `Click` event.

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

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

#### BY THE WAY

---

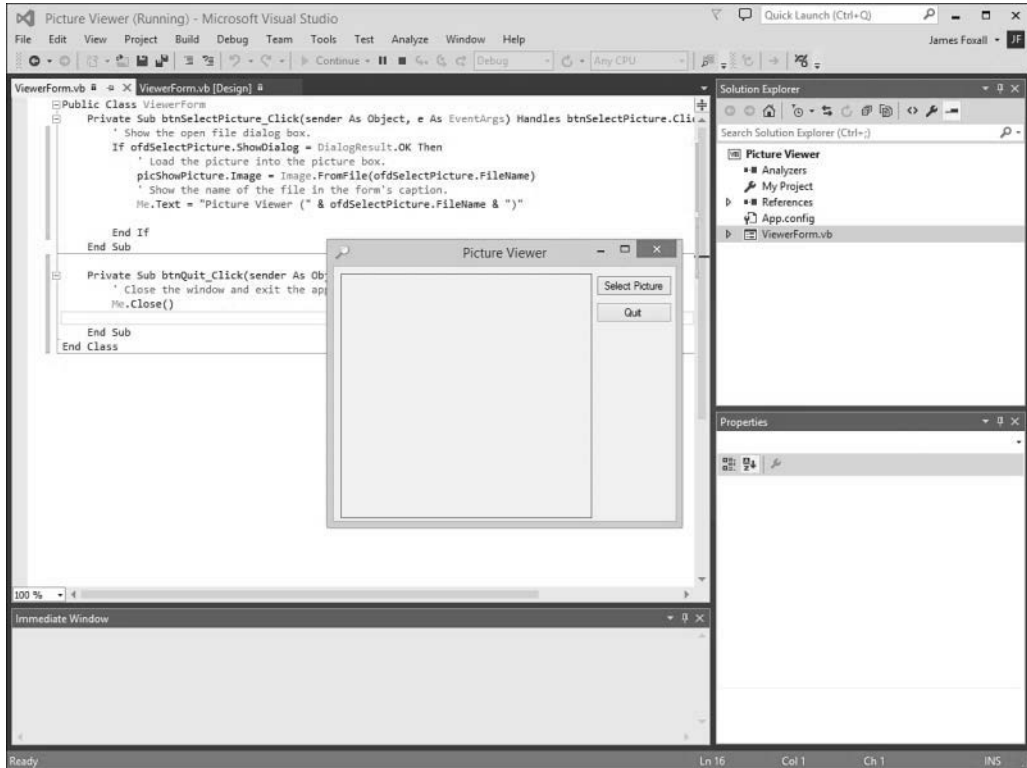
The `Me.Close()` statement closes the current form. When the last loaded form in a program is closed, the application shuts itself down—completely. As you build more robust applications, you'll probably want to execute all kinds of cleanup routines before terminating an application, but for this example,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

---

## Running a Project

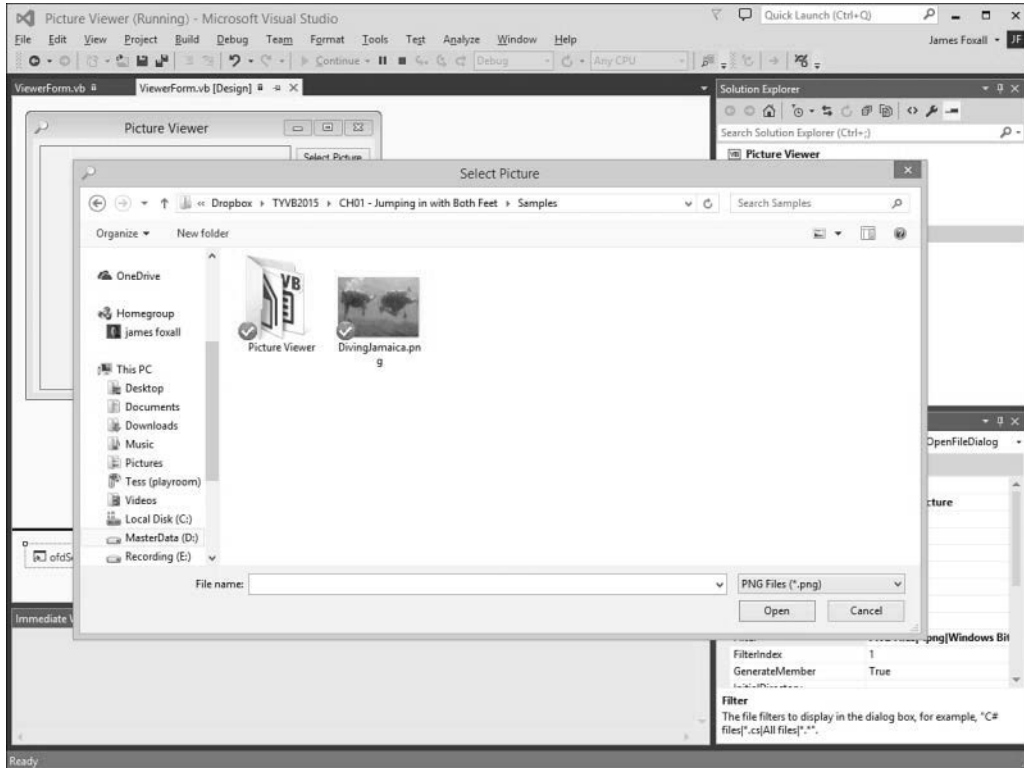
Your application is now complete. Click the `Save All` button on the toolbar (the button with two floppy disks), and then run your program by pressing `F5`. You can also run the program by clicking the button on the toolbar that looks like a right-facing triangle and resembles the `Play` button on a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

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

**FIGURE 1.14**

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

You are now running your program as though it were a standalone application running on another user's machine; what you see is exactly what users would see if they ran the program (without the Visual Studio 2015 design environment in the background, of course). Click the Select Picture button to display the Select Picture dialog box, shown in Figure 1.15. Use this dialog box to locate a picture file. When you've found a file, double-click it, or click once to select it and then click Open. The selected picture is then displayed in the picture box, as shown in Figure 1.16.

**FIGURE 1.15**

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

---

#### BY THE WAY

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

---

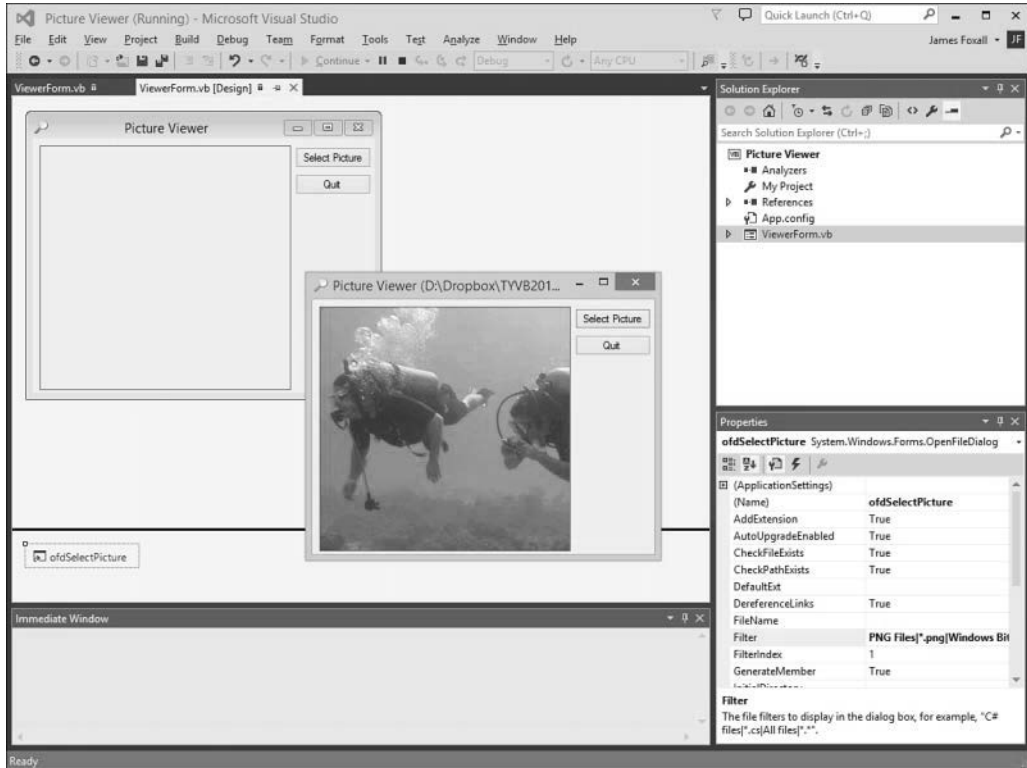
---

#### BY THE WAY

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

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

---



**FIGURE 1.16**  
Displaying pictures is easy if you know just a few techniques.

## Summary

That's it! You've just created a bona fide Visual Basic program. You've used the toolbox to build an interface with which users can interact with your program, and you've written code in strategic event handlers to empower your program to do things. These are the basics of application development in Visual Basic. Even the most complicated programs are built using this fundamental approach: You build the interface and add code to make the application do things. Of course, writing code to do things *exactly* the way you want things done is where the process can get complicated, but you're on your way.

If you take a close look at the organization of the hours in this book, you'll see that I start out by teaching you the Visual Basic (Visual Studio .NET) environment. I then move on to building an interface, and later I teach you about writing code. This organization is deliberate. You might be eager to jump in and start writing serious code, but writing code is only part of the



equation—don't forget the word *Visual* in Visual Basic. As you progress through the hours, you'll build a solid foundation of development skills.

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

## Q&A

**Q. Can I show bitmaps of file types other than BMP, JPG, and PNG?**

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

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

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

## Workshop

### Quiz

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

### Answers

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

## Exercises

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

*This page intentionally left blank*

# Index

## Symbols

- & (ampersand), 300
- ' (apostrophe), 21, 345
- \* (asterisk), 71, 171
- = (equals sign), 65
- + (plus), 300
- \_ (underscore), 244

## A

- Abort, DialogResult, 396
- Accept buttons, creating, 173-174
- AcceptButton property, 173-174
- AcceptsReturn property, 169
- accessing
  - Help, 60
  - object's events, 91-93
  - Windows Registry, with My.Computer.Registry, 460-463
- Add() method, 182-183, 206

## adding

- comments, to code, 344-345
- controls
  - to forms, 13-14
  - to forms with toolbox, 41-43
- controls to forms, 131-133
- to Date/Time, 305-306
- images to backgrounds, forms, 113-116
- invisible controls, to forms, 18-19
  - items, to lists, 182-183
  - list items with code, 206
  - to enhanced lists, 204-206
- nodes to Tree View control, 208-210
- PageSetupDialog controls, 508-509
- Print button, to forms, 506-508
- Print Preview button, to forms, 506-508
- PrintDocument controls, 508-509

- PrintPreviewDialog control, 508-509
- project files, 55-56
- scrollbars, to text boxes, 169
- Send Email toolbar buttons, 530-531
- toolbar buttons, with Items collection, 230-233
- visible controls to forms, 15-17
- addition, performing, 292**
- Add/Remove Programs dialog box, 553**
- adjusting grid, granularity, 135**
- ADO.NET, 484-485**
  - closing connections to data sources, 491
  - connecting to databases, 485-491
  - connection object,
    - ConnectionString property, 487
  - connection strings, building, 487-489
  - ConnectionString property, 489-490
  - creating new records, 499-500
  - DataAdapter, creating, 492-493
  - DataRow, referencing fields in, 494-496
  - DataTable, 491
    - creating/populating, 493-494
  - deleting records, 500-501
  - editing records, 498-499
  - navigating records, 496-498
  - running database examples, 502
  - System.Data, 484
- advanced breakpoint features, 357**
- advanced options for ClickOnce programs, 556**
- aligning controls, 140-141**
- alignment, text alignment, text boxes, 166-167**
- ampersand (&), 300**
- Anchor property, 145**
- anchoring controls, 143-149**
- And, 297**
- anticipated exceptions, 364-367**
- apostrophe ('), 21, 345**
- appearance of forms, changing, 109**
  - adding images to background, 113-116
    - buttons, 117-119
  - assigning icons, 116-117
  - background color, 111-113
  - borders, 119-121
  - controlling size, 121-122
  - displaying text on title bars, 110-111
- applications, uninstalling, 553-555**
- arguments, 252**
- Arial font, 424**
- arithmetic operations**
  - equalities, comparing, 295-296
  - performing, 291-292
    - addition, 292
    - division, 293
    - exponentiation, 293
    - modulus arithmetic, 293
    - multiplication, 292-293
    - negation, 292
    - order of operator precedence, 294-295
    - subtraction, 292
- arrays, 259, 273**
  - dimensioning, 273
  - multidimensional arrays, creating, 274-276
  - referencing array variables, 273-274
  - two-dimensional arrays, 274
- As Integer, 247, 252**
- assigning**
  - icons, to forms, 116-117
  - shortcut keys, to menu items, 227-229
- asterisk (\*), 71, 171**
- Attachment, .NET classes, 530**
- attributes**
  - file attribute flags, 449
  - of files, getting, 449
  - object attributes, as properties, 376-378
- Auto Hide, 39**
- auto hiding, design windows, 39**
- AutoScroll, 153**
- AutoScrollMargin, 153**
- AutoScrollMinSize, 153**
- autosizing, controls, 143-149**
- avoiding**
  - infinite recursion, 255-256
  - recursive events, 90

**B**

**BackColor** property, 111-113

**background color, changing on forms, 111-113**

**BaseDirectory()**, 477

**benefits of constants, 264**

**binary data, 53**

**binding object references to variables, 382**

- early-binding, 384-385
- late-binding, 382-384

**bitmaps, Graphics object, 415-416**

**block scope, 277**

**Boolean, 260, 263**

**Boolean logic, 291, 296-297**

**Boolean operators, 297**

- And, 297
- Not, 297-298
- Or, 298
- Xor, 298

**borders, customizing (forms), 119-121**

**branching within procedures, GoTo, 324**

**breakpoints, 349-351**

- advanced breakpoint features, 357

**breaking only when hit a certain number of times, 359**

- stopping code execution, 358

**BringToFront** method, 151

**browsing files, writing code for, 20-22**

**brushes, 424**

**btnQuit** button, 23

**build errors, 346-349**

**Button** controls, 14, 322

**buttons**

- Accept buttons, creating, 173-174
- Cancel buttons, creating, 174-175
- creating, 172-173
- message boxes, 393-396
  - determining which button is clicked, 396-397
- Print button, adding, 506-508
- Print Preview button, adding, 506-508
- Quit button, 23
- removing, 250-251
- Save All button, 10
- Send Email toolbar buttons, adding, 530-531

**Byte, 260****ByVal, 253****C****calling**

- code procedures, 248-251
- passing parameters, 252-254
- Function, 250

**Cancel, DialogResult, 396****Cancel buttons, creating, 174-175****CancelButton** property, 174-175**Case Else, 319**

**Case statements, evaluating, 320-321**

**casting, data from one data type to another, 262-263**

**casting downward, 262****casting upward, 262****Catch, 361-363**

- anticipated exceptions, 364

**CenterParent, 126****CenterScreen, 126****ChangePageSettings(), 520****changing**

- form's name, 108
- printer and page settings, 519-521
- properties, 44-46
- size, of forms, 11-12

**Char, 260****check boxes, yes/no options, 175****checked menu items, creating, 220-222****CheckState** property, 175**child forms, 156-157****circles, drawing, 423****class interfaces, 374****class modules, 54, 240****classes, 372**

- comparing with standard modules, 373

- encapsulating data and code, 372-373

- exposing object attributes as properties, 376-378

- instantiating objects from, 381-382

- object interfaces, creating, 374-376

- System.Random class, 425

**ClassesRoot, 460****Clear() method, 76, 185, 207, 423****clearing**

- drawing surfaces, 423

- lists, 185

- nodes, Tree View control, 211

**Click, 406****Click events, 20, 93, 172**

**ClickOnce, 545-546**

- creating applications, with Publish Wizard, 547-551
- Picture Viewer ClickOnce install program, testing, 552
- setting advanced options for programs, 556
- clients, 373
- interactions with objects, 375

**CLng(), 355****Close button, adding, to forms, 117-119****Close() method, 127****closing**

- connections to data sources, ADO.NET, 491
- loops, with Next statement, 330-331

**COBOL, 561****code**

- adding list items, 206
- comments, adding, 344-345
- for email, testing, 541-542
- encapsulating in classes, 372-373
- manipulating, List View control, 206-207
- referencing properties, 65-66
- removing, list items, 207
- writing
  - for browsing files, 20-22
  - with procedures, 58-59
  - to retrieve file properties, 450-451
  - to send emails, 537-541
  - terminating programs, 23-24

**code labels, 325****code procedures**

- calling, 248-251

- passing parameters, 252-254
- writing, 242-243
  - declaring procedures that don't return values, 243-247
  - declaring procedures that return values, 247-248

**collections, 79-81**

- Controls collection, 79-81

**color drop-down list, 47****color properties, 46-49****colors, system colors, 417-419****columns in enhanced lists, List View control, 203****Combo Box control, 188-190****combo boxes, creating drop-down lists, 188-190****comments, 21**

- adding to code, 344-345
- creating, 345

**common language runtime, .NET Framework, 560****common type system, .NET Framework, 563****comparing**

- classes, with standard modules, 373
- equalities, 295-296

**comparison operators, 295****components, 53-54**

- class modules, 54
- distributable components, 2forms, 54
- modules, 54
- of For statement, 330
- user controls, 54

**concatenating, strings, of text, 299-300****concatenation, 81****condition, 332****conditions, stopping code execution, 358****connecting to databases, ADO.NET, 485-491****connection object,****ConnectionString property, 487****connection strings, building, 487-489****ConnectionString property, 487-490****constants, 259, 263-265**

- benefits of, 264
- creating, 265
- defining, 264

**constructs, If...Then, 313-315****containers**

- Group Box controls, 176-178
- radio buttons, 178-180

**Context Menu Strip control, 225****context menus, implementing, 225-227****continuing, looping before Next is reached, 332****control box buttons, adding to forms, 117-119****Control objects, 7****controlling, size, of forms, 121-122****controls, 131, 163**

- adding to forms, 13-14, 131-133
- with toolbox, 41-43

**aligning, 140-141****Button controls, 14****buttons. See buttons****containers, radio buttons, 178-180****Context Menu Strip control, 225**

Graphics object, creating, 414

Group Box controls, 176-178

Image List control, 200-202

invisible controls, adding to forms, 18-19

Label controls, 97

- displaying static text, 163-164

layering, 151

List Box control, 180

List View control

- building enhanced lists. *See* enhanced lists

manipulating with code, 206-207

manipulating, 133

- aligning controls, 140-141
- anchoring, 143-149
- autosizing, 143-149
- with grid (size and snap), 133-136
- selecting groups of controls, 138-140
- setting property values for groups of controls, 142
- sizing controls, 142
- with snap lines, 136
- spacing evenly, 142

Menu Strip control, 216

Open File Dialog control, 193

OpenFileDialog controls, 14. *See also* OpenFileDialog controls

PageSetupDialog controls, 508-509

Panel, 176-178

PictureBox controls, 14

placing on group boxes, 178

PrintDocument controls, adding, 508-509

PrintPreviewDialog control, adding, 508-509

sizing, 142

Status Bar control, 235-237

Tab control, 197-200

tbrMainToolbar control, 233

Text Box control, 172

Textbox control, 89

Timer control, 194

ToolStrip control, 229-230

Tree View control, 201

- adding nodes, 208-210
- clearing all nodes, 211
- creating hierarchical lists, 207-208
- removing nodes, 211
- user controls, 54
- visible controls, adding to forms, 15-17

**Controls collection, 79-81**

**Copy() method, 444-445**

**copying files, File object, 444-445**

**CreateGraphics, 75**

**CreateGraphics(), 414**

**CreateSubKey() method, 461**

**CurrentConfig, 460**

**currently viewed images, printing, 514-516**

**CurrentUser, 460**

**custom dialog boxes, creating, 398-401**

**Custom Dialog Example, creating, 398-401**

**custom object events, 375**

**customizing**

- design windows, 35-36
- forms, 109
  - adding buttons, 117-119
  - adding images to background, 113-116

- assigning icons, 116-117
- background color, 111-113
- borders, 119-121
- controlling size, 121-122
- displaying text on title bars, 110-111

## D

### **DashStyle, 416-417**

**data, encapsulating, in classes, 372-373**

**data sources, closing connections to, 491**

### **data types, 260**

- casting data from data type to another, 262-263

Date, 304-305

- determining, 260

- guidelines for, 261-262

Object, 262

Time, 309

- type conversion functions, 262

- value ranges, 260

### **data typing, 260**

- enforced variable declaration, 269

### **DataAdapter, 484**

- ADO.NET, 492-493

**databases, connecting to, ADO.NET, 485-491**

### **DataReader, 484**

**DataRow, ADO.NET, referencing fields in, 494-496**

### **DataSet, 484**



**DataTable, 484**

- ADO.NET, 491
  - creating new records, 499-500
  - creating/populating, 493-494
  - deleting records, 500-501
  - editing records, 498-499
  - navigating records, 496-498

**Date, 260****Date data type, 304-305****date information, getting for files, 448****DateAdd(), 305-307****DateDiff(), 307****Date/Time**

- adding to/subtracting from, 305-306
- determining intervals between, 307
- determining whether values are dates, 309
- formatting, 308
- retrieving
  - current system, 309
  - parts of dates, 307-308

**debugging**

- breakpoints, breaking only when hit a certain number of times, 359
- comments, adding to code, 344-345
- errors, 346-349
- Picture Viewer project, Windows Registry, 467-470
- sending messages to Output window using trace points, 360

- stopping code execution under specific conditions, 358

**debugging tools, 349**

- advanced breakpoint features, 357
- breakpoints, 349-351
  - Immediate window, 351-356

**Debug.WriteLine() method, 356, 360****Decimal, 260****decisions**

- Elseif, 317-318
- GoTo, 324
- If...Then, 313-315
  - executing code when expression is false, 316
  - nesting, 318-319

**Declarations section, modules, 279****declaring**

- procedures that don't return values, 243-247
- procedures that return values, 247-248
- variables, 58, 266-267
- variables of static scope, 281-282

**default values, 267****defaultresponse parameter, 402****Define Color dialog box, 48****Delete() method, 79, 446-447****deleting**

- event procedures, 99
- files, File object, 446-447
- menu items, 220
- records, ADO.NET, 500-501
- Registry keys, 461-462

**deliberate recursion, 256****design time, manipulating, items, 181****design windows**

- auto hiding, 39
- customizing, 35-36
- docked windows, sizing, 38
- docking, 37-39
- floating, 36
- showing/hiding, 36

**designing interfaces**

- invisible controls, 18-19
- visible controls, 15-17

**Destination variable, 513****dialog boxes**

- creating, custom dialog boxes, 398-401
- Define Color dialog box, 48
- New Project dialog box, 4
- Save Project dialog box, 10
- Send Email dialog box, creating, 532-536
- tabbed dialog boxes, creating, 197-200

**DialogResult, MessageBox.Show(), 396****Dim, 267****Dim statement, 277**

- dimensioning arrays, 273
- variables, creating new objects, 386

**directories, manipulating with Directory object, 452-453****Directory object, manipulating, directories, 452-453****display position, specifying for forms, 126-127****displaying**

- lists with list boxes, 180-181
- log files, Picture Viewer project, 477-479

messages, with MessageBox.  
   show() function, 391-393  
 options from Windows  
   Registry, 464-465  
 static texts, with Label con-  
   trols, 163-164  
 text on title bars, forms,  
   110-111  
**distributable components, 2. See**  
   *also programs*  
**division, performing, 293**  
**docking, design windows, 37-39**  
**documents**  
   previewing, 517-518  
   printing, 510-516  
     creating PrintImage proce-  
     dures, 512-513  
     currently viewed images,  
     514-516  
**DoesSourceFileExist() method,**  
   **444**  
**Do.Loop, 336**  
   creating, 336  
   examples, 338-341  
   ending, 336-337  
**Do.Loop structure, 277**  
**Double, 260**  
**double-clicking, 7**  
   adding controls to toolbox,  
   132  
   in Solution Explorer, 52  
**dragging controls from toolbox,**  
   **132**  
**DrawBorder(), 476**  
**DrawEllipse() method, 423**  
**DrawImage() method, 430**  
**drawing**  
   brushes, 424  
   clearing drawing surface, 423

  controls, to add to forms,  
     132-133  
   to forms, 425  
   pens, 416-417  
     rectangles, 421-422  
     shapes, 422  
     circles, 423  
     ellipses, 423  
     lines, 422  
     rectangles, 423  
   system colors, 417-419  
   text, 423-425  
**drawing surfaces, clearing, 423**  
**DrawLine() method, 422**  
**DrawRectangle() method, 76-77,**  
   **423**  
**DrawString() method, 423-425**  
**drop-down lists, creating with**  
   **combo boxes, 188-190**  
**drop-down menus, creating, for**  
   **toolbar buttons, 234**  
**dynamic applications, creating, 22**

## E

**early-binding, object variables,**  
   **384-385**  
**editing records, ADO.NET,**  
   **498-499**  
**ellipses, drawing, 423**  
**Else, 316**  
**Elseif, 317-318**  
**emails, sending**  
   adding a Send Email toolbar  
   button, 530-531  
   classes used to send, 530  
   creating Send Email dialog  
   box, 532-536  
   testing code, 541-542  
   writing code for, 537-541  
**Enabled property, 168**  
**encapsulating data and code in**  
   **classes, 372-373**  
**End Function, 254**  
**End If statements, 22, 314**  
**End Sub, 243, 245, 254**  
**ending Do.Loop, 336-337**  
**enforced variable declaration,**  
   **data typing, 269**  
**enhanced lists, building with List**  
   **View control, 202-203**  
   adding list items, 204-206  
   creating columns, 203  
**enumerations, 510**  
   creating, 511  
**equalities, comparing, 295-296**  
**equals sign (=), 65**  
**error handlers**  
   anticipated exceptions,  
   364-367  
   exceptions, 363-364  
   writing, Try...Catch...Finally,  
   360-363  
**Error icon, 395**  
**Error List window, 272**  
**errors**  
   build errors, 346-349  
   runtime errors, 346-349  
**evaluating**  
   Case statements, If...Then,  
   320-321  
   expressions, Select Case,  
   319  
**event declarations, 93-94**  
**event handlers, creating, 98-103**  
**event names, keeping current,**  
   **103**

**event parameters, 95-97****event procedures**

- creating, 92
- deleting, 99

**event projects, building**

- event handlers, 98-103
- user interfaces, 97-98

**event-driven language, 20****event-driven programming, 87-88**

- accessing object's events, 91-93
- event names, 103
- event parameters, 95-97
- recursive events, avoiding, 90
- triggering events, 88

**events**

- Click events, 93
- custom object events, 375
- object's events, accessing, 91-93
- Paint event, 429-430
- recursive events, avoiding, 90
- text boxes, 172
- triggering, 88
  - by objects, 89
  - by operating systems, 90
  - through user interaction, 88-89

**exceptions, 346**

- anticipated exceptions, 364-367
- causing, 365
- error handlers, 363-364

**Exclamation, MessageBoxIcon, 393****Exists() method, 443-444****Exit statements, 254****exiting**

- loops early, For...Next, 332
- procedures, 254

**explicit variable declaration, 269-270****exponentiation, performing, 293****expression argument, 304**

- executing code when expression is false, 316

**expressions, variables, 268****extensions, .vb, 53****F****False, executing code when expression is false, 316****file attribute flags, 449****file filters, creating in OpenFileDialog controls, 439-440****File object, manipulating files, 443**

- copying files, 444-445
- deleting files, 446-447
- determining whether files exist, 443-444
- moving files, 445-446
- retrieving file properties, 447-451

**file operations**

- OpenFileDialog controls, 435-439
  - creating file filters, 439-440
- SaveFileDialog controls, 441-443

**file properties**

- retrieving, 447-451
- writing code for retrieving, 450-451

**FileName property, 439****files**

- attributes, getting for files, 449
- browsing, writing code for, 20-22
- copying with File object, 444-445
- deleting with File object, 446-447
- getting date and time information, 448
- manipulating with File object, 443
  - copying files, 444-445
  - deleting files, 446-447
  - determining whether files exist, 443-444
  - moving files, 445-446
  - retrieving file properties, 447-451
- moving with File object, 445-446
- SaveFileDialog controls, 435-436

**Filter property, 19, 439****FilterIndex property, 440****Finally, 361-363****findtext argument, 304****floating, design windows, 36****Font object, 424****For statement, initiating loops, 330****Form object, 7****Format(), 308****Format16bppGrayScale, 415**

**Format16bppRgb555, 415**

**Format24bppRgb, 415**

**formatting, Date/Time, 308**

**FormBorderStyle property, 119-121**

**forms, 54, 107**

adding

controls with toolbox, 41-43

invisible controls, 18-19

Print button, 506-508

Print Preview button, 506-508

visible controls, 15-17

buttons. *See* buttons

changing appearance, 109

adding buttons, 117-119

adding images to background, 113-116

assigning icons, 116-117

background color, 111-113

borders, 119-121

controlling size, 121-122

displaying text on title bars, 110-111

changing names, 108

changing size, 11-12

check boxes, 175

child forms, 156-157

controls

adding, 13-14, 131-133

layering, 151

manipulating. *See* manipulating

displaying in normal, maximized or minimized state, 124-125

drawing to, 425

Graphics object, creating, 414

icons, 11

MDI forms, creating, 154-158

modality, 123-124

nonmodal windows, creating, 151

parent forms, 156-158

preventing from appearing on taskbars, 127

removing images, 116

scrollable forms, creating, 152-153

showing, 122-123

specifying initial display position, 126-127

startup forms, 158-159

tab order, creating, 149-151

text boxes, 164-166

events, 172

limiting number of characters, 170-171

multiline text boxes, 167-168

password fields, 171-172

scrollbars, 169

text alignment, 166-167

Text property, 10

transparent forms, creating, 151-152

unloading, 127-128

**For...Next, 329**

closing with Next statement, 330-331

continuing looping before Next is reached, 332

creating, 332-334

exiting loops early, 332

initiating loops using For, 330

specifying increment value with Step, 331-332

**forward references, 7**

**Friend, 282**

**Function, 58, 247-248**

calling, 250

**functions, 242**

classes, 375

DateAdd(), 305-306, 307DateDiff(), 307

declaring, 247-248

exposing as methods, 381Format(), 308

IsDate(), 309

IsNumeric(), 314-315

LTrim(), 303

Replace(), 304

RTrim(), 303

strings

Instr(), 302-303

Len(), 300

Microsoft.VisualBasic.Left(), 300-301

Microsoft.VisualBasic.Mid(), 301-302

Microsoft.VisualBasic.Right(), 301

Trim(), 303-304

type conversion functions, casting data from data type to another, 262

## G

**garbage collection, .NET Framework, 564**

**GDI (graphical device interface), 414**

**Get construct, creating readable properties, 378**

**GetAttributes(), 449, 451**

**GetCreationTime, 448**

**GetLastAccessTime**, 448  
**GetLastWriteTime**, 448  
**GetSetting()**, 460  
**GetValue()** method, 462  
**ghost forms**, 151-152  
**global scope**, 279-280  
**Gmail**, 529  
**good messages, creating with**  
    **MessageBox.show()** function, 397  
**GoTo**, 324  
**granularity, adjusting grid**, 135  
**graphical device interface. See**  
    **GDI (graphical device interface)**  
**graphics**  
    drawing. *See* drawing  
    forms, drawing to, 425  
    pens, 416-417  
    Persisting Graphics project,  
    creating, 425-431  
    rectangles, 421-422  
    shapes. *See* shapes  
    system colors, 417-419  
**Graphics object**, 413-414  
    creating  
        for bitmaps, 415-416  
        for forms/controls, 414  
        drawing to forms, 425  
**grid, manipulating, controls**,  
    133-136  
**GridSize**, 134-135  
**Group Box controls**, 176-178  
**group boxes**, 176-178  
    placing controls on, 178  
**groups of controls, selecting**,  
    138-140  
**guidelines for, data types**,  
    261-262

## H

**Handles**, 103  
**Height property**, 12  
**Help**, 59-60  
**hiding**  
    design windows, 36  
    toolbars, 40-41  
**hierarchical lists, creating, with**  
    **Tree View control**, 207-208  
**HKEY\_CLASSES\_ROOT**, 458  
**HKEY\_CURRENT\_CONFIG**, 458  
**HKEY\_CURRENT\_USER**, 458, 461  
**HKEY\_LOCAL\_MACHINE**, 458  
**HKEY\_USERS**, 458

## I

**Icon parameter, message boxes**,  
    393  
**icons**  
    assigning icons, to forms,  
    116-117  
    Error icon, 395  
    giving to forms, 11  
    message boxes, 393-396  
    Question icon, 395  
**IDE (integrated development envi-  
 ronment)**, 2, 6-7, 360  
    adding, controls to forms with  
    toolbox, 41-43  
    customizing, design windows,  
    35-36  
**If test**, 359  
**If...Then**, 277, 295, 313-316  
    Elseif, 317-318  
    evaluating, Case statements,  
    320-321

    nesting, 318-319  
    Select Case, 323-324  
        building examples,  
        321-323  
**Ignore, DialogResult**, 396  
**IL (Intermediate Language)**,  
    560-562  
**Image List control**, 200-203  
**images**  
    adding to backgrounds, forms,  
    113-116  
    removing from forms, 116  
    scaling to fit a page, 522-527  
    storing in Image List controls,  
    200-202  
**ImageSize property**, 201  
**Immediate window, debugging**  
    tools, 351-356  
**implementing context menus**,  
    225-227  
**Imports**, 486  
**increment value, specifying incre-  
 ment value with Step**, 331-332  
**infinite recursion, avoiding**,  
    255-256  
**Inflate()** method, 422  
**Information, MessageBoxIcon**,  
    393  
**initial display position, specifying**  
    for forms, 126-127  
**InitialDirectory property**, 439  
**initializing options variables**,  
    283-286  
**InputBox()**, 401-404  
**Insert()** method, 183  
**instantiating objects from classes**,  
    381-382  
**Instr()**, 302-303  
**Integer**, 260

integrated development environment. *See* IDE (integrated development environment)

#### interfaces

creating for drawing project, 74

designing

adding invisible controls to forms, 18-19

adding visible controls to forms, 15-17

Intermediate Language (IL), 560-562

Interval property, 89, 194

intervals, between Date/Time, 307

Invalidate() method, 431

invisible controls, adding to forms, 18-19

invoking, methods, 73

IsDate(), 309

IsNumeric(), 314-315

itemname, 463

#### items

adding to lists, 182-183

manipulating at design time, list boxes, 181

manipulating at runtime, list boxes, 182-187

removing from lists, 183-184

Items collection, 181

adding toolbar buttons, 230-233

Items Collection Editor, 233, 506

Items property, 188

## J

JITter (just-in-time compiler), 561

## K

key values, Registry keys, 462-463

Keyboard Example project, 405

keyboard input, 404-405

keyboards, user interaction, 404-406

KeyChar property, 406

KeyDown, 404

keypath, 463

KeyPress, 404, 406

keys, Registry keys. *See also* Registry keys

KeyUp, 404

#### keywords

Function, 58

Handles, 103

reserved keywords, 267

Sub, 58

To, 320

Until, 337

While, 337

## L

Label controls, 97

displaying, static text, 163-164

language runtime, 560

LargeImageList property, 203

late-binding, object variables, 382-384

layering, controls, 151

LayoutMode, 134

layouts, multiple layouts, 40

Len(), 300

lifetime of, objects, 387

limiting number of characters in text boxes, 170-171

lines, drawing, 422

List Box control, 180

list boxes, 180-181

items, manipulating  
at design time, 181  
at runtime, 182-187

#### list items

adding  
with code, 206  
to enhanced lists,  
204-206  
removing, 207  
with code, 207

#### List View control

building enhanced lists,  
202-203  
adding list items, 204-206  
creating columns, 203  
manipulating, using code,  
206-207

#### lists

clearing, 185  
displaying with list boxes,  
180-181  
drop-down lists, creating with  
combo boxes, 188-190  
enhanced lists. *See*  
enhanced lists  
hierarchical lists, creating with  
Tree View control, 207-208  
items  
adding, 182-183  
removing, 183-184

- retrieving information about
  - selected items in lists, 185-187
- sorting, 187

**literal values, passing to variables, 268**

**local scope, 278**

**LocalMachine, 460**

**Location property, 179**

**log files, Picture Viewer project**

- creating, 474-477
- displaying, 477-479
- testing, 479

**Long, 260**

**looping**

- Do.Loop, 336
  - creating, 336
  - creating examples, 338-341
  - ending, 336-337
- For.Next, 329
  - closing with Next statement, 330-331
  - continuing looping before Next is reached, 332
  - creating, 332-334
  - exiting loops early, 332
  - initiating using For, 330
  - specifying increment value with Step, 331-332

**LTrim(), 303**

## M

**MailMessage, 532**

- .NET classes, 530

**managed code, 560**

**managing projects, 50**

- with Solution Explorer, 50-52

**manipulating**

- controls, 133
  - aligning controls, 140-141
  - anchoring, 143-149
  - autosizing, 143-149
  - with grid (size and snap), 133-136
  - selecting groups of controls, 138-140
  - setting property values for groups of controls, 142
  - sizing, 142
  - with snap lines, 136
  - spacing evenly, 142
- directories, with Directory object, 452-453
- files with File object, 443
  - copying files, 444-445
  - deleting files, 446-447
  - determining whether files exist, 443-444
  - moving files, 445-446
  - retrieving file properties, 447-451
- items

- at design time, list boxes, 181
- at runtime, listboxes, 182-187
- List View control, with code, 206-207

**items**

- at design time, list boxes, 181
- at runtime, listboxes, 182-187

- List View control, with code, 206-207

**Manual, 126**

**math, arithmetic operations. See arithmetic operations**

**Maximize button, adding, to forms, 117-119**

**maximized state, forms, 124-125**

**MaximumSize property, 121-122**

**MaxLength property, 170-171**

**MDI forms, creating, 154-158**

**MDIs (multiple document interfaces), 131**

**Me.Close(), 24**

**menu items**

- assigning shortcut keys, 227-229
- checked menu items, creating, 220-222
- creating for top-level menus, 219-220
- deleting, 220
- moving, 220

**Menu Strip control, 216**

**menus**

- checked menu items, creating, 220-222
- context menus, implementing, 225-227
- menu items, assigning shortcut keys, 227-229
- programming menus, 223-225
- top-level menus
  - creating, 216-219
  - creating menu items for, 219-220

**message boxes**

- buttons, determining which button is clicked, 396-397
- creating good messages, 397
- displaying with MessageBox.show() function, 391-393
- specifying buttons and icons, 393-396

**MessageBoxButtons, 393**

**MessageBoxIcon, 393-394**

**MessageBox.Show(), 59, 81, 186**

- creating good messages, 397

- DialogResult, 396
  - displaying messages, 391-393
  - messages, displaying with**
    - MessageBox.show() function, 391-393
  - metadata, 563**
  - method dynamism, 73**
  - methods, 72**
    - Add() method, 182-183, 206
    - BringToFront method, 151
    - classes, 375
    - Clear() method, 185, 207, 423
    - Close() method, 127
    - Copy() method, 444-445
    - CreateSubKey() method, 461
    - Debug.WriteLine() method, 356
    - Delete() method, 79, 446-447
    - DoesSourceFileExist() method, 444
    - DrawEllipse() method, 423
    - DrawImage() method, 430
    - DrawLine() method, 422
    - DrawRectangle() method, 76, 77, 423
    - DrawString() method, 423-425
    - Exists() method, 443-444
    - functions as, 381
    - GetAttributes() method, 451
    - GetValue() method, 462
    - Inflate() method, 422
    - Insert() method, 183
    - Invalidate() method, 431
    - MessageBox.Show(), 81
    - Move() method, 445-446
    - NewRow() method, 484
    - Remove() method, 183-184, 207, 211
    - SelectNextControl() method, 150
    - SendToBack() method, 151
    - SetValue() method, 462
    - Show() method, 123
    - ShowDialog() method, 21, 400, 440-441
    - triggering, 73
  - Microsoft Intermediate Language (IL), 560-562**
  - Microsoft.VisualBasic namespace, 562**
  - Microsoft.VisualBasic.Left(), 300-301**
  - Microsoft.VisualBasic.Mid(), 301-302**
  - Microsoft.VisualBasic.Right(), 301**
  - Mid(), 301-302**
  - Minimize button, adding, to forms, 117-119**
  - minimized state, forms, 124-125**
  - MinimumSize property, 121-122**
  - modal forms, 123-124**
  - modifying Picture Viewer project**
    - to use Registry, 464-470
    - to use text files, 474-479
  - module-level scope, 278-279**
  - modules, 54**
    - class modules, 240
    - creating, 239-241
    - Declarations section, 279
      - declaring procedures that don't return values, 243-247
    - standard modules, 240
      - comparing with classes, 373
    - creating, 241
  - modulus arithmetic, performing, 293**
  - mouse clicks, double-clicking, 7**
  - mouse events, 406-409**
  - mouse input, 406**
  - MouseDown, 93, 95, 172, 406**
  - MouseEnter, 406**
  - MouseEventArgs, 95**
  - MouseHover, 406**
  - MouseLeave, 100, 406**
  - MouseMove, 103, 172, 406**
  - MousePaint project, 407-409**
  - MouseUp, 172, 406**
  - Move() method, 445-446**
  - moving**
    - files, File object, 445-446
    - menu items, 220
  - MsgBox() function, 393**
  - multidimensional arrays, creating, 274-276**
  - Multiline property, 167, 169**
  - multiline text boxes, 167-168**
  - multiple document interfaces.**
    - See MDIs (multiple document interfaces)
  - multiplication, performing, 292-293**
  - My.Computer.Registry, accessing Windows Registry, 460-463**
- ## N
- name conflicts, scope, 280-281**
  - Name property, 8-9**
  - names, changing form's name, 108**
  - namespace scope, 279-280**
  - namespaces, .NET Framework, 562-563**



**naming**

- objects, 8-9
- suffixes, 9
- variables, 286

**naming collisions, 562****navigating records, ADO.NET, 496-498****negation, performing, 292****nesting If...Then constructs, 318-319****.NET Framework, 2, 556, 559**

- classes, for sending email, 530
- common language runtime, 560
- common type system, 563
- garbage collection, 564
- IL (Intermediate Language), 560-562
- namespaces, 562-563

**new features, multiple layouts, 40****New Project dialog box, 4****NewRow() method, 484****Next, continuing looping before**

Next is reached, For...Next, 332

**Next statement, closing loops, 330-331****No, DialogResult, 396****nodes**

- adding to Tree View control, 208-210
- clearing, Tree View control, 211
- removing, Tree View control, 211

**None**

- DialogResult, 396
- MessageBoxIcon, 393

**nonmodal forms, 123****nonmodal windows, creating, 151****normal state, forms, 124-125****Not, 297-298****Notepad, 54****O****Object, 260, 262****object attributes, as properties, 376-378****Object Browser, 82-83****object interfaces, creating, 374-376****object references**

- binding to variables, 382
- early-binding, 384-385
- late-binding, 382-384
- releasing, 387

**object-based code, writing, 74-78****object-oriented programming, 64****objects, 7-8, 64**

- building simple object projects, 73-74
- creating interfaces, 74
- testing, 78
- writing object-based code, 74-78

**collections, 79-81**

creating when dimensioning variables, 386

Directory object, manipulating directories, 452-453

events, triggering, 89

Font object, 424

forms, 7

Graphics. See Graphics object  
instantiating from classes, 381-382

lifetime of, 387

methods, 72

    dynamism, 73

    triggering, 73

naming, 8-9

    suffixes, 9

properties, 8, 64

    changing, 44-46

    color properties, 46-49

    referencing in code, 65-66

    viewing, 44

    viewing property descriptions, 49-50

    working with, 67-72

Registry object, 460

Text property, 10

**objFile.Close(), 471****objFile.Dispose(), 471****objGraphics, 76, 416****OK, DialogResult, 396****On Error statements, 360****Opacity property, 151-152****Open File dialog box**

file filters, creating, 439-440

OpenFileDialog controls, 436-439

showing, 440-441

**OpenFileDialog controls, 14,**

**15-17, 193, 435-439**

creating, file filters, 439-440

**opening existing projects from Start page, 34****OpenPicture() function, 474****OpenPicture() procedure, 245, 475****operating systems, triggering, events, 90****operators. See also Boolean operators****Option Infer, 272**

options variables, initializing,  
283-286  
Or, 297, 298  
order of operator precedence,  
294-295

## P

page settings, changing, 519-521  
pages, tabs, 197  
PageSetupDialog controls, adding,  
508-509  
Paint event, 90, 429-430  
Panel control, 176-178  
parameters  
    ConnectionString property,  
    487  
    defaultresponse parameter,  
    402  
    event parameters, 95-97  
    itemname, 463  
    keypath, 463  
    passing, 252-254  
parent forms, 156-158  
parentheses, 244  
passing  
    literal values to variables, 268  
    parameters, code procedures,  
    252-254  
password fields, text boxes,  
171-172  
PasswordChar property, 171-172  
pens, 416-417  
performing arithmetic operations,  
291-292  
    addition, 292  
    division, 293  
    exponentiation, 293  
    modulus arithmetic, 293

multiplication, 292-293  
negation, 292  
order of operator precedence,  
294-295  
subtraction, 292

**Persisting Graphics project, creat-  
ing, 425-431**

**Picture Viewer ClickOnce install  
program, testing, 552**

**Picture Viewer project**

adding  
    PageSetupDialog controls,  
    508-509  
    Print button, 506-508  
    Print Preview button,  
    506-508  
    PrintDocument controls,  
    508-509  
    PrintPreviewDialog control,  
    508-509

log files

    creating, 474-477  
    displaying, 477-479  
    testing, 479

modifying, to use Registry,  
464-470

modifying to use text files,  
474-479

sending email

    adding a Send Email tool-  
    bar button, 530-531  
    creating Send Email dialog  
    box, 532-536  
    testing email code,  
    541-542  
    writing code for, 537-541

tabbed dialog boxes, adding,  
197-200

testing and debugging,  
467-470

working with, objects and  
properties, 67-72

**PictureBox controls, 14**

**pictures, storing, in Image List  
controls, 200-202**

**pixelformat, 415**

**pixels, 12**

**plus (+), 300**

**Pointer, 133**

**populating DataTable, ADO.NET,  
493-494**

**preventing forms from appearing  
on taskbars, 127**

**previewing documents, 517-518**

**Print button, adding, to forms,  
506-508**

**Print Preview button, adding, to  
forms, 506-508**

**Print Preview window, 518**

**PrintDocument controls, 509**

    adding, 508-509

**printer settings, changing,  
519-521**

**PrintImage procedures, creating,  
512-513**

**printing**

    documents, 510-516

        creating PrintImage proce-  
        dures, 512-513

        printing currently viewed  
        page, 514-516

    printer and page settings,  
    changing, 519-521

    scaling images to fit a page,  
    522-527

**PrintPage event, 523-524**

**PrintPreviewDialog control, 517**

    adding, 508-509

**Private, 94, 512**

**procedure-level scope, 278**

**procedures. See also code procedures**

- exiting, 254
- functions, 242
- hooking up, 253
- infinite recursion, avoiding, 255-256
- PrintImage procedures, creating, 512-513
- ResizeToPrintableArea procedure, 524
- subroutines, 242
- writing, 242-243
  - declaring procedures that don't return values, 243-247
  - declaring procedures that return values, 247-248
  - functional units of code, 58-59

**program interaction, 391**

- InputBox(), 401-404
- keyboards, 404-406
- MessageBox.Show(). See MessageBox.Show()
- mouse events, 406-409

**programming menus, 223-225****programming toolbars, 233-234****programs**

- defined, 52
- terminating, code for, 23-24

**project files, adding/removing, 55-57****project properties, 54-55****projects, 2**

- creating new, 3-6
  - Start page, 32-34
- defined, 52

- managing, 50
  - with Solution Explorer, 50-52
- opening, from Start page, 34
- running, 24-26
- saving, 10

**properties, 8, 64**

- AcceptButton property, 173-174
- AcceptsReturn property, 169
- Anchor property, 145
- BackColor property, 111-113
- CancelButton property, 174-175
- changing, 44-46
- CheckState property, 175
- classes, 375
- color properties, 46-49
- ConnectionString property, 487-490
- Enabled property, 168
- FileName property, 439
- Filter property, 19, 439FilterIndex property, 440
- FormBorderStyle property, 119-121
- Height property, 12
- ImageSize property, 201
- InitialDirectory property, 439
- Interval property, 89, 194
- Items property, 188
- KeyChar property, 406
- LargeImageList property, 203
- Location property, 179
- MaximumSize property, 121-122
- MaxLength property, 170-171
- MinimumSize property, 121-122

- Multiline property, 167, 169
- Name property, 8-9
- object attributes as, 376-378
- Opacity property, 151-152
- PasswordChar property, 171-172
- project properties, 54-55
- radio buttons, 179
- readable properties, creating with Get construct, 378
- read-only properties, 66
  - creating, 380
- referencing in code, 65-66
- Registry object, 460
- ScrollBars property, 169
- SelectedIndex properties, 187
- SelectedItem property, 185-187, 207
- SelectionMode property, 187
- ShowGrid property, 136
- ShowInTaskbar property, 127
- SizingGrip property, 237
- StartPosition property, 126-127
- SubItems property, 205
- TabIndex properties, 149
- Text property, 10, 111
- TextAlign property, 166-167
- Title property, 19, 439
- Toolbox window, 151
- TransparentColor property, 202
- viewing, 44
  - property descriptions, 49-50
- Visible property, 128
- Width property, 12
- WindowsState property, 124
- working with, 67-72

writable properties, creating  
with Set construct, 379

write-only properties, creating,  
380

**Properties pane, Properties window, 44**

**Properties window, 7, 43**  
Description section, 49-50  
resolution, 7

**property descriptions, viewing, 49-50**

**property values, setting for groups of controls, 142**

**Public Sub, 243**

**Publish Wizard, 556**  
creating ClickOnce applications, 547-551

## Q

**Question icon, 395**

## R

**radio buttons, 178-180**  
properties, 179  
Select Case, 323

**random numbers, 425**

**readable properties, creating with Get construct, 378**

**reading, text files, Windows Registry, 472-473**

**read-only properties, 66**  
creating, 380

**ReadToEnd(), 478**

**records**  
creating new (ADO.NET), 499-500  
deleting (ADO.NET), 500-501  
editing (ADO.NET), 498-499  
navigating, (ADO.NET), 496-498

**rectangles, 421-422**  
drawing, 423

**recursive events, avoiding, 90**

**recursive loops, 256**

**referencing**  
array variables, 273-274  
fields, in DataRow (ADO.NET), 494-496

**REG\_BINARY, 458**

**REG\_EXPAND\_SZ, 458**

**REG\_MULTI\_SZ, 458**

**REG\_SZ, 458**

**Registry. See Windows Registry**

**Registry keys**  
creating, 460-461  
deleting, 461-462  
key values, 462-463

**Registry object, 460**  
top-node properties, 460

**releasing, object references, 387**

**Remove() method, 183-184, 207, 211**

**removing**  
applications, 553-555  
buttons, 250-251  
images, from forms, 116  
items, from lists, 183-184  
list items, 207  
with code, 207  
nodes, Tree View control, 211  
project files, 57

**renaming variables, 286**

**Replace(), 304**

**replacetext argument, 304**

**replacing text within strings, 304**

**reserved keywords, 267**

**ResizeToPrintableArea procedure, 524**

**resolution, Properties window, 7**

**retrieving**  
current system, Date/Time, 309  
file properties, 447-451  
code for, 450-451  
information about selected items in lists, 185-187  
parts of dates, 307-308

**Retry, DialogResult, 396**

**Return, 247**

**RTrim(), 303**

**Run mode, 24**  
runningdatabase examples, ADO.NET, 502  
projects, 24-26

**runtime, 560**  
manipulating, items, 182-187

**runtime errors, 346-349**

## S

**Save All button, 10**

**Save File dialog box, creating, 441-443**

**Save Project dialog box, 10**

**SaveFileDialog controls, 435-436, 441-443**

**SaveSetting(), 460**

**saving**

- options to Windows Registry, 465
- projects, 10

**SByte, 260****scaling images to fit a page, 522-527****scope, 259, 276-277**

- block scope, 277
- declaring variables of static scope, 281-282
- global scope, 279-280
- local scope, 278
- module-level scope, 278-279
- name conflicts, 280-281
- procedure-level scope, 278

**scope designator, 244****scrollable forms, creating, 152-153****scrollbars, adding, to text boxes, 169****ScrollBars property, 169****SDI (single-document interface), 154****Select Case, 319, 323-324**

- building examples, 321-323
- evaluating, 320-321

**SelectedIndex properties, 187****SelectedItem property, 185-187, 207****SelectedItems collection, 207****selecting, groups of controls, 138-140****SelectionMode property, 187****SelectNextControl() method, 150****Send Email dialog box, creating, 532-536****Send Email toolbar buttons, adding, 530-531****sending****emails**

- adding a Send Email toolbar button, 530-531
- classes, 530
- creating Send Email dialog box, 532-536
- testing code, 541-542
- writing code for, 537-541
- messages to Output window using tracepoints, 360

**SendToBack() method, 151****servers, 373****Set construct, writable properties, creating, 379****SetValue() method, 462****shapes, drawing, 422**

- circles, 423
- ellipses, 423
- lines, 422
- rectangles, 423

**Short, 260****shortcut keys, assigning to menu items, 227-229****Show() method, 123****ShowCurrentRecord() method, 496****ShowDialog() method, 21, 400, 440-441****ShowGrid, 134****ShowGrid property, 136****showing**

- design windows, 36
- forms, 122-123
- Open File dialog box, 440-441
- toolbars, 40-41

**ShowInTaskbar property, 127****simple object projects, building, 73-74**

- creating interfaces, 74
- testing, 78
- writing object-based code, 74-78

**Single, 260****single-document interface (SDI), 154****size, of forms, controlling, 121-122****sizing**

- docked windows, 38
- forms, 11-12

**sizing handles, 139****SizingGrip property, 237****SmtplibClient, .NET classes, 530****snap lines, manipulating, controls, 136****SnapToGrid, 134****Solution Explorer**

- double-clicking, 52
- managing, projects, 50-52

**Solution Explorer window, 50-51**

- solutions, 2, 53
- defined, 53

**sorting lists, 187****source files, 11****spaces, 244**

- trimming from strings, 303-304

**spacing controls, evenly, 142****spaghetti code, 324****SqlConnection, 484****SqlDataAdapter, 492-493****StackOverflow exception, 90****standard modules, 240**

- comparing with classes, 373
- creating, 241

**Start page, 31**

- creating new projects, 32-34
- opening existing projects, 34

**starting Visual Studio 2015, 2-3****StartPosition property, 126-127****startup forms, 158-159****state of forms, displaying in normal, maximized or minimized state, 124-125****static scope, declaring variables, 281-282****static texts, displaying with Label controls, 163-164****static variables, 282****Status Bar control, 235-237****status bars, creating, 235-237****StatusStrip, 236****StreamReader, 472-473****Step, specifying increment value, 331-332****Stop, MessageBoxIcon, 393****stopping, code execution under specific conditions, 358****storing**

- pictures in Image List controls, 200-202
- values in variables, 58

**StreamWriter, 470-472****strict typing, 269**

- variables, 270-272

**String, 260****strings, 298-299**

- concatenating, text, 299-300
- functions
  - Instr(), 302-303
  - Len(), 300
  - Microsoft.VisualBasic.Left(), 300-301

**Microsoft.VisualBasic.**

- Mid(), 301-302

**Microsoft.VisualBasic.**

- Right(), 301

- replacing text within, 304
- trimming spaces from, 303-304

**Structured Exception Handling project, creating, 360-361****structures, 277****Sub, 58, 94, 243, 248****SubItems property, 205****subkeys, 461****subroutines, 242****subtracting from Date/Time, 305-306****subtraction, performing, 292****suffixes, naming objects, 9****system colors, 417-419****System namespace, 562****SystemColors, 75****System.Data, ADO.NET, 484****System.Data namespace, 562****System.Diagnostics, 562****System.Drawing namespace, 562****System.IO, 443****System.IO namespace, 562****System.IO.Directory, 452-453****System.IO.File object, 447, 449, 451****System.Net namespace, 562****System.Net.Mail, 530****System.Random class, 425****System.Security namespace, 562****System.Web namespace, 562****System.Windows.Forms namespace, 562****System.Xml namespace, 562****T****Tab control, 197-200****tab order, creating for forms, 149-151****tabbed dialog boxes, creating, 197-200****TabIndex properties, 149****TabPage Collection Editor, 197****tabs, collections, pages, 197****taskbars, preventing forms from appearing on, 127****tbrMainToolBar control, 233****Templates list, 32****terminating programs, code for, 23-24****testing**

- email code, 541-542
- Picture Viewer ClickOnce install program, 552
- Picture Viewer logs, 479
- Picture Viewer project, Windows Registry, 467-470
- simple object projects, 78

**text**

- concatenating, strings, 299-300
- displaying on title bars, forms, 110-111
- drawing, 423-425
- replacing in strings, 304

**text alignment, text boxes, 166-167****Text Box control, 172****text boxes, 164-166**

- events, 172
- limiting number of characters, 170-171
- multiline text boxes, 167-168

- password fields, 171-172
  - scrollbars, adding, 169
  - text alignment, 166-167
  - text files, 53, 457**
    - modifying, Picture Viewer project to use, 474-479
    - reading, Windows Registry, 472-473
    - writing, Windows Registry, 470-472
  - Text property, 10, 111**
  - TextAlign property, 166-167**
  - Textbox control, 89**
  - TextChanged, 172**
  - texts, static texts, displaying with Label controls, 163-164
  - time. See Date/Time
  - Time data type, 309**
  - time information, getting for files, 448
  - TimeOfDay(), 196**
  - Timer control, 194**
  - timers, creating, 193-196
  - title bars, displaying text, forms, 110-111
  - Title property, 19, 439**
  - To, 320**
  - toolbar buttons**
    - adding with Items collection, 230-233
    - creating drop-down menus for, 234
  - toolbar items, 229**
  - toolbars, 40, 229**
    - drop-down menus for toolbar buttons creating, 234
    - programming toolbars, 233-234
    - showing/hiding, 40-41
    - toolbar buttons, adding with Items collection, 230-233
  - toolbox, 14**
    - adding controls to forms, 41-43
    - double-clicking to add controls to forms, 132
    - dragging to add controls to forms, 132
  - Toolbox window, 6**
    - tools, debugging tools, 349
    - advanced breakpoint features, 357
    - breakpoints, 349-351
    - Immediate window, 351-356
  - toolbarstrip, 229**
  - ToolStrip control, 229-230**
  - top-level menus, creating, 216-219**
    - menu items, 219-220
  - TopMost property, nonmodal windows, 151**
  - tracepoints, 360**
  - transparent forms, creating, 151-152**
  - TransparentColor property, 202**
  - Tree View control, 201**
    - clearing nodes, 211
    - creating hierarchical lists, 207-208
    - nodes, adding, 208-210
    - removing nodes, 211
  - triggering**
    - events, 88
      - by objects, 89
      - by operating systems, 90
      - through user interaction, 88-89
    - methods, 73
  - Trim(), 303-304**
  - trimming, spaces from strings, 303-304**
  - Try, 361, 363**
  - Try...Catch...Finally, 360-363**
  - Try...End Try structures, 367**
  - two-dimensional arrays, 274**
  - type conversion functions, casting data from data type to another, 262**
- ## U
- UInteger, 260**
  - ULong, 260**
  - underscore (\_), 244**
  - uninstalling applications, 553-555**
  - unloading forms, 127-128**
  - Until, 337**
  - Update(), 498**
  - user controls, 54**
  - user interaction**
    - InputBox(), 401-404
    - keyboards, 404-406
    - mouse events, 406-409
    - triggering events, 88-89
  - user interfaces, creating, 97-98**
  - Users, Registry object, 460**
  - UShort, 260**
- ## V
- value data types, Windows Registry, 458**
  - value items, 462**
  - value ranges, data types, 260**

**values**

- determining if values are dates, 309
- increment value, specifying
  - increment value with Step, 331-332
- literal values, passing to variables, 268
- storing in variables, 58

**variables, 252, 259, 266**

- array variables, referencing, 273-274
- binding object references to, 382
  - early-binding, 384-385
  - late-binding, 382-384
- creating, 282-283
- creating new objects when
  - dimensioning variables, 386
- declaring, 58, 266-267
  - of static scope, 281-282
- Destination variable, 513
- explicit variable declaration, 269-270
- options variables, initializing, 283-286
- passing literal values to, 268
- renaming, 286
- static variables, 282
- storing values, 58
- strict typing, 270-272
- using in expressions, 268

**.vb, 53****View Detail window, 354****ViewerForm.vb\*, 23****viewing**

- properties, 44
- property descriptions, 49-50

**visible controls, adding to forms, 15-17****Visible property, 128****Visual Basic 2015 Start page, 31**

- creating new projects, 32-34
- opening existing projects, 34

**Visual Studio 2015, starting, 2-3****W****Warning, MessageBoxIcon, 393****While, 337****Width property, 12****Window Color and Appearance dialog box, 418-419****windows**

- CLng(), 355
- Immediate window. See Immediate window
- nonmodal windows, creating, 151
- Print Preview window, 518
- Properties window, 7, 43
- Solution Explorer window, 50-51
- Toolbox window, 6
- View Detail window, 354

**Windows Forms Application, creating, 4-5****Windows Forms Application projects, creating, 343-344****Windows Registry, 457-458**

- accessing with My.Computer.Registry, 460-463
- displaying options from, 464-465
- modifying, Picture Viewer project to use, 464-470

## nodes, 458

Picture Viewer project, testing and debugging, 467-470

reading text files, 472-473

saving options to, 465

structure of, 458-460

using options stored in Registry, 465-466

value data types, 458

writing text files, 470-472

**WindowsDefaultBounds, 126****WindowsDefaultLocation, 126****WindowState property, 124****With, 439****wizards, Publish Wizard, 556**

creating ClickOnce applications, 547-551

**writable properties, creating with Set construct, 379****Write(), 471-472****WriteLine(), 471-472****write-only properties, creating, 380****writing**

## code

- for browsing files, 20-22
- with procedures, 58-59
- to retrieve file properties, 450-451
- to send emails, 537-541
- terminating programs, 23-24
- code procedures, 242-243
  - declaring procedures that don't return values, 243-247
  - declaring procedures that return values, 247-248



error handlers

    anticipated exceptions,  
    364-367

    exceptions, 363-364

    Try.Catch.Finally, 360-363

object-based code, simple

    object projects, 74-78

text files, Windows Registry,

    470-472

## **X**

XML files, 53

Xor, 297, 298

## **Y-Z**

Yes, DialogResult, 396

yes/no options, check boxes, 175