

SAMS

# FLUENT VISUAL BASIC®

REBECCA M. RIORDAN



SAMS

# FLUENT VISUAL BASIC®

REBECCA M. RIORDAN



ASSOCIATE PUBLISHER  
Greg Wiegand

SIGNING EDITOR  
Neil Rowe

MANAGING EDITOR  
Kristy Hart

PROJECT EDITOR  
Andy Beaster

INDEXER  
Cheryl Lenser

PROOFREADER  
Karen Gill

TECHNICAL EDITOR  
John Hardesty

PUBLISHING COORDINATOR  
Cindy Teeters

COVER DESIGNER  
Gary Adair

COMPOSITION  
Rebecca Riordan

FLUENT VISUAL BASIC®  
Copyright © 2011 by Rebecca Riordan

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: 9780672335808

ISBN-10: 0672335808

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

Printed in the United States of America

First Printing November 2011

#### 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.

The Windlass Lowercase and Brandywine fonts are copyrights of the Scriptorium foundry, [www.fontcraft.com](http://www.fontcraft.com).

#### 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.

#### BULK SALES

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

U.S. Corporate and Government Sales  
1-800-382-3419  
[corpsales@pearsontechgroup.com](mailto:corpsales@pearsontechgroup.com)

For sales outside of the U.S., please contact

International Sales  
[international@pearson.com](mailto:international@pearson.com)

# ACKNOWLEDGEMENTS

Yes, I know it says "Rebecca M. Riordan" on the cover, but that's not really true. Without the assistance of some amazing people, this book would never have made it out of my head, much less into your hands. So, in order of appearance, I would like to thank:

Neil Rowe, my editor, who took a chance on a very different way of writing computer tutorials. Without Neil's leap of faith, Fluent Learning would never have happened. My technical reviewers, David Sceppa, Eric Weinburger and John Hardesty, who collectively caught an embarrassing number of code typos and I-knew-what-I-meant obscurities. Finally, my copy editor, Karen Gill, who not only made sure the language in the book resembles English, but also expressed an unexpected and greatly appreciated enjoyment in the project. (Any remaining errors and infelicities are, of course, my responsibility.)

Jake von Slatt of The Steampunk Workshop ([steampunkworkshop.com](http://steampunkworkshop.com)), Samantha Wright ([samantha-wright.deviantart.com](http://samantha-wright.deviantart.com)) and Mindbloom ([mindbloom.com](http://mindbloom.com)) were all gracious enough to allow me to use their images. These are all seriously cool people, folks. I can't urge you strongly enough to go explore their sites.

## GETTING STARTED

### INTRODUCTION . . . . . I

- Fluent Learning Because
- This book isn't for everyone
- What you'll learn
- What you'll need
- How it works

### APPLICATION DEVELOPMENT . . . . . 9

- The development process
- System design
- Creating executables

### THE .NET PLATFORM . . . . . 43

- .NET Components
- Say hello
- Say what?

### THE VISUAL STUDIO UI . . . . . 67

- Solutions, projects and stuff
- Take control
- Get some help

### TESTING & DEPLOYMENT . . . . . 93

- Errors & exceptions
- Deployment

Find out what this whole "being a programmer" thing is all about and how to use the tools you'll need to build applications.

Learn how to speak Visual Basic. It's a language, much like English, Spanish or Latin, only simpler.

## THE LANGUAGE

### PART 1: NOUNS . . . . . 121

- Statements
- Declared elements
- Comments
- Directives & Attributes

### PART 2: TRANSITIVE VERBS . . . 155

- Literal expressions
- Object expressions

### PART 3: INTRANSITIVE VERBS . . 181

- Control of flow commands
- Exception handling commands

Discover the secret to efficient programming: The best code is the code you don't have to write yourself.

## THE .NET FRAMEWORK LIBRARY

### CLASSES IN THE .NET FRAMEWORK . 221

- The Class Designer
- Class definitions
- Fields & properties
- Methods

### OTHER FRAMEWORK TYPES . . . . . 269

- Structures
- Enumerations
- Interfaces
- Working with types

### THE CLASS LIBRARY, PART 1 . . . 305

- Namespaces
- The Object Browser
- Numeric data
- Character data
- Times & dates

### THE CLASS LIBRARY, PART 2 . . . 349

- Arrays
- Specialized Sets
- Generics

# CONTENTS

Put all you've learned to good use by learning how to use Microsoft's latest and greatest interface platform.

Stand on the shoulders of the experts by learning the best programming practices and how to implement them.

## BEST PRACTICE

### OOA & D ..... 381

- Type relationships
- OOP principles
- Type modifiers

### PROGRAMMING PRINCIPLES... 425

- The Single Responsibility Principle
- The Open/Closed Principle
- The Liskov Substitution Principle
- The Law of Demeter

### PATTERNS ..... 457

- The Strategy Pattern
- The Observer Pattern
- Architectural Patterns

## WPF

### XAML ..... 501

- Fundamentals
- WPF types
- XAML & Visual Basic

### WPF CONTROLS ..... 531

- WPF panels
- Control classes
- Content controls
- Items controls

### DEPENDENCY PROPERTIES... 591

- The basics
- Creating dependency properties

### WPF INTERACTIONS..... 627

- Routed events
- WPF Commands

### WPF GRAPHICS ..... 669

- Color
- Brushes
- Pens
- Typography
- Effects

### RESOURCES ..... 719

- Resource dictionaries
- Styles
- Property triggers
- Event triggers

### TEMPLATES ..... 765

- Building controls
- Building control templates
- The VisualStateManager

### WPF BINDING ..... 797

- Creating bindings
- Binding to collections
- Working with collections

# TELL US WHAT YOU THINK!

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.

As a Executive Editor for Sams, I welcome your comments. You can fax, email, or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books stronger.

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

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

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

Fax: 317-428-3310

Mail: Neil Rowe, Executive Editor  
Sams Publishing  
800 East 96th Street

Indianapolis, IN 46240 USA



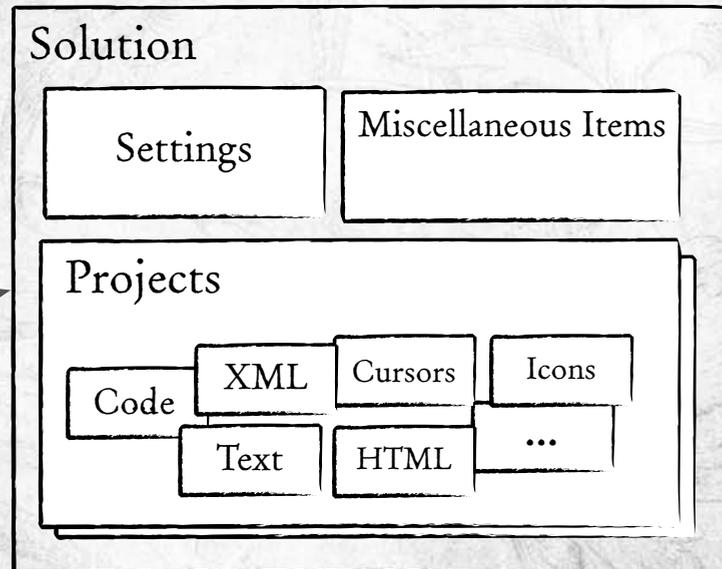
# THE VISUAL STUDIO UI



In the last chapter you wrote your first program and saw the basics of the Visual Studio user interface (UI). Now it's time to look at these steps in more detail. We'll start by looking at how Visual Studio helps you manage a development project with Solutions and Projects, and then take a closer look at the UI and how to configure it to suit the way you work.

As a programmer, you'll spend a lot of time in the Code Editor, and so will we. We'll look at the basic text editing functions it provides and also at Intellisense and the Visual Studio help system.

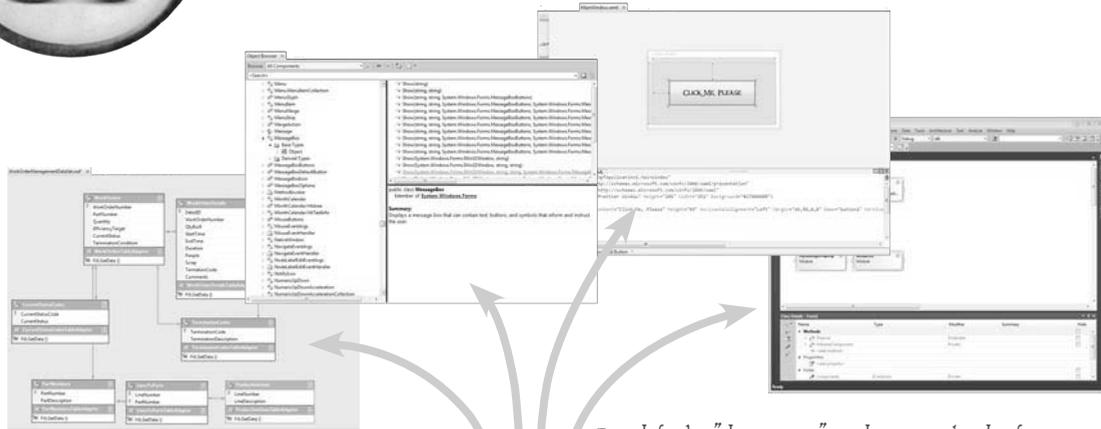
Solutions contain Projects, Solution Settings that control how the application will be compiled and run, and Solution Items that aren't part of a specific project.



Solutions can also contain other files that aren't included in the application but are available from the Solution Explorer when the Solution is open.



# IN A NUTSHELL



The contents of the Toolbox change to reflect the kind of document you're working on.

By default, "documents" and certain kinds of dialogs are displayed as tabs in the central portion of the Visual Studio window.

The document outline window is a great way to navigate hierarchical documents like XML or XHTML

The Properties window is a quick and easy way to change the attributes of something that's selected in a designer. You can also use it as a shortcut to create and manage event handlers.

The Data Source window is like an explorer window for external data like a relational database.

Some Visual Studio Designers can display multiple panes. The WPF Designer shown here, for example, shows XAML and a design surface you can use for drag-and-drop.

Like the Windows Explorer, the Solution Explorer helps you navigate & manage the files and folders associated with your project.



# TASK LIST

A craftsman is master of his tools. As a programmer, your primary tool is Visual Studio, and in this chapter we'll begin the process of mastery by examining its user interface in detail.



## SOLUTIONS, PROJECTS & STUFF

It's convenient to think of application development like writing an essay or book: You do some research, prepare an outline, and then produce a final document. Unfortunately, the development process isn't that neat. (Neither is writing, of course, at least not the way I do it.) Most development projects don't even have a single output that's equivalent to that essay. So we'll start this chapter by looking at the way Visual Studio uses Solutions, Projects and Solution Items to manage all the bits and pieces that you'll actually be working with.



## TAKE CONTROL

I bet you've changed your Windows desktop. If you're like most people, you've added widgets to the sidebar, created some shortcuts, and rearranged the Start menu. All those little changes just make life a little easier by putting the tools you use all the time close to hand. Visual Studio does a pretty good job of arranging the user interface to accommodate general programming, but you'll benefit from making the same sorts of customizations to its workspace as you made to the Windows desktop, so the next thing we'll do is learn how to do just that.



## GET SOME HELP

In the last chapter we saw an example of Intellisense when we were able to pick the `MessageBox.Show` command from a drop-down list. In this chapter, we'll look at Intellisense in more detail, along with some of the special error-checking capabilities that the Visual Studio Editor provides.



# SOLUTIONS, PROJECTS...

Solutions are like filing cabinets that hold and manage Projects and other files. Easy enough in principle, but what exactly does that mean? Why would you have more than one Project? What are these "other files", and what exactly does "manage" mean? Let's look at some examples of the kinds of files you might include in a Solution.



## PRIMARY UI PROJECTS

You'll usually have a project (sometimes more than one) that contains the forms that comprise your application user interface.



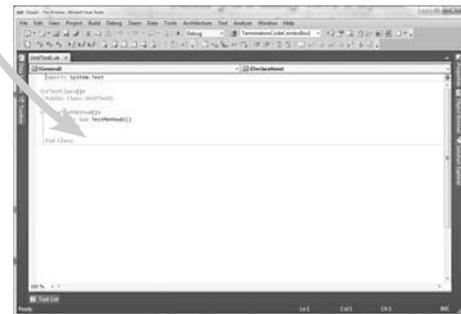
## DATASET PROJECTS

If your application references a data source, you'll usually have a separate project that handles the data interface.



## UTILITY CLASSES

In cooking, you often need to translate between teaspoons, tablespoons and cups. The classes that do that conversion might be useful in other applications, so it makes sense to put those in a separate project that we can reference when we need them.



## TEST PROJECTS

Visual Studio 2010 provides great support for a technique called Test-Driven Development. If you adopt this approach, you'll need projects that contain your tests.

# ...AND STUFF

Projects have multiple files, as well. You'll have code and designer files, of course, you've already seen that, but Visual Studio will also allow you to associate other files with the project, just to keep them handy.



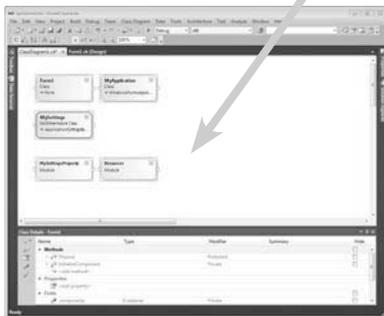
## DESIGNER FILES

These are the files that are created by the Visual Studio Designers. They're text files, but if you mess with them outside the Designer, your changes might get overwritten the next time you use the Designer, so you'll typically leave them alone.

## DESIGN DOCUMENTS

You can also include documents like this class diagram or even specifications.

These files aren't part of the project code.

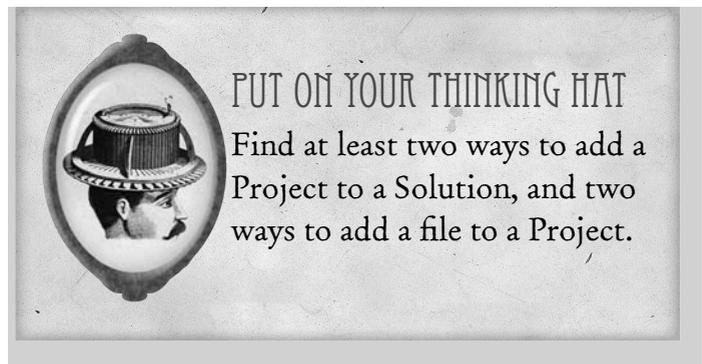


## SOURCE FILES

These are the files that contain your code. They have the same name as the files the Designer creates, with the extension ".vb".

## RESOURCES

If your project includes things like custom cursors or icons, these are separate files in the Project.

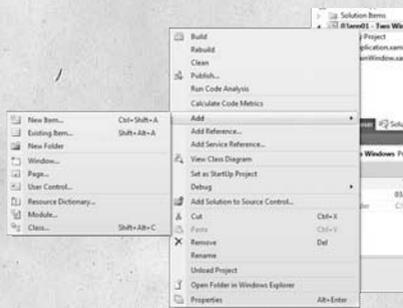




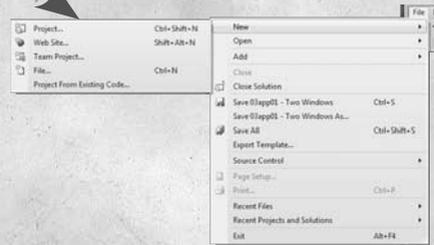
## HOW'D YOU DO?

The problem was to figure out how to add Solutions and files....

## TO ADD A PROJECT TO A SOLUTION



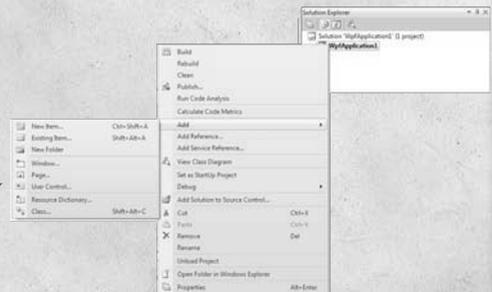
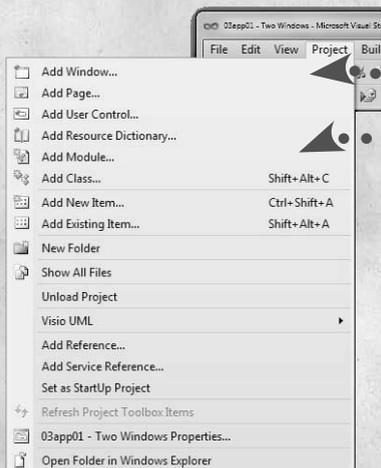
The New Project item can be found on the File menu and on the context menu displayed when you right-click the Solution name in the Solution Explorer.



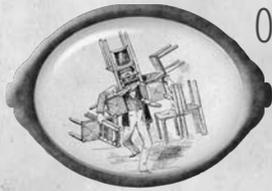
You can also add existing Projects, which is handy if you're reusing utilities or custom widgets, or exclude a project that you've added by accident.

## TO ADD A FILE TO A PROJECT

You can add new and existing items from the Solution Explorer by right-clicking the Project name.



The Project menu provides specific options for the most common types of Projects, or you can choose Add New Item... to display the New Item dialog.



## ON YOUR OWN

You know how to add an item to a Project, and you know how to configure a simple WPF window, so let's put those two things together. Change your Hello, World application to display a window instead of a MessageBox:

- 1 Add a new window to the Project. Accept the default name of Window1.
- 2 Drag a Label from the Toolbox to the window design surface, and configure the window and label properties however you like. Have some fun! You can't hurt anything.
- 3 Display the code for the original form (NOT the one you just created). If the form is open, you can double-click the button or press F7. If the form isn't open, you can right-click the form name in the Solution Explorer and choose Show Code.
- 4 Delete the line that displays the MessageBox, and replace it with the following:

Declare a variable called "newWin". VARIABLES are just a name for a piece of memory that can store information or objects of a certain type. This one stores a Window.

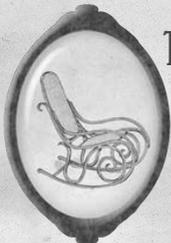
- You'll use variables a lot when you program.
- We'll look at them in Chapter 5.

```
Dim newWin As New Window1()  
newWin.ShowDialog()
```

The variable is INITIALIZED (given an initial value) with an instance of Window1, the Window you just created.

- This line calls the ShowDialog() method of the window. METHODS are something an object can do. We'll look at them in detail in Chapter 8.

- 5 Run the application by pressing F5, and then click the button on the first form.



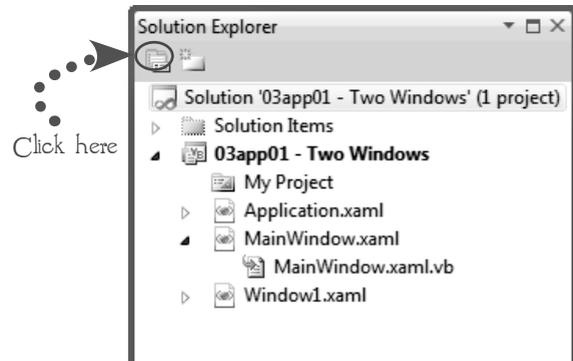
## TAKE A BREAK

Why don't you take a quick break before we move on to controlling the way Solutions and Projects behave by setting their properties.

# SOLUTION AND...

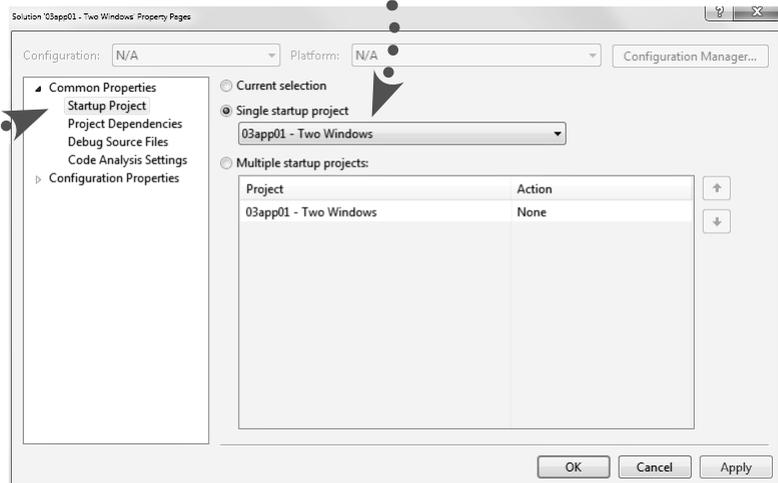
When you created and modified your windows on the design surface, you saw that you could control the appearance of the widgets by setting their properties. Well, Solution and Projects have properties, as well. We'll need some of these as we move through the book, so let's get started by looking at how to display the property dialogs.

The easiest way to display the Solution Properties dialog is to select the Solution in the Solution Explorer and click the Properties button on the Solution Explorer toolbar, but you can also choose Properties Page from the View menu.



Most of the Solution properties are managed by Visual Studio, and you only need to change them in unusual circumstances, but you'll often need to specify the Startup Project whenever you have a Solution that contains multiple Projects. The file specified as the Startup is the one that Visual Studio will run when you press F5 or run the final application.

By default, Visual Studio sets the first project you add to the Solution as the Startup Project. You can change that by choosing a different Project from the combobox.



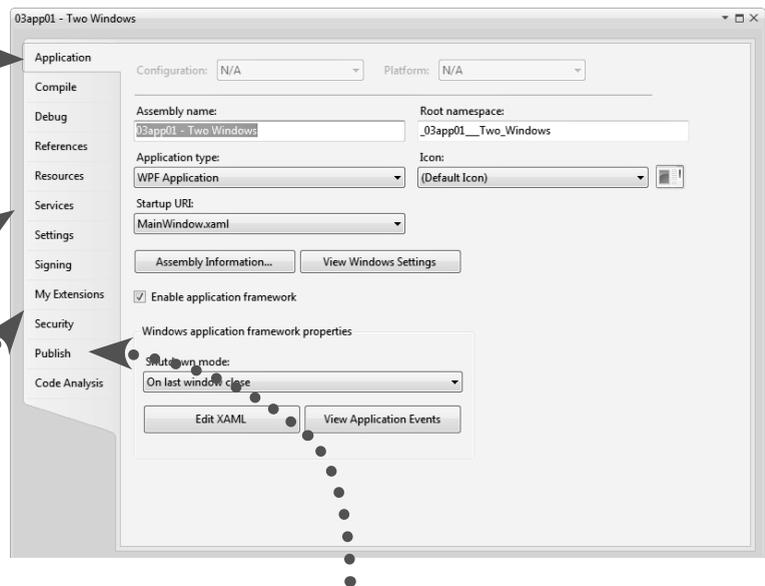
# ...PROJECT PROPERTIES

Solution properties display in a dialog box, but Visual Studio has a designer (called the Project Designer) for project properties that displays in a tab. You can display Project Designer by clicking the Solution Explorer toolbar button when a Project is selected or by choosing <ProjectName> Properties... from the Project menu when the Project is selected, or by right-clicking the Project name in the Solution Explorer and choosing Properties, or by selecting the Solution name in the Solution Explorer and pressing Alt+Enter. And probably by doing some other things that I haven't discovered yet...

The Application tab will change depending on the type of Project you choose, but it always controls the type of application and how it is compiled.

Resources and Settings are things like icons and strings that are included in the executable. We'll use these tabs in just a minute to create an icon for our application.

Signing and Security help you secure your application and its users from bad people and bad software. Security is an important issue, but it's also a huge one, so we won't be talking about it in any detail.



The Publish tab is used to deploy your application using ClickOnce. We'll talk about that in the next chapter.

! Not all of these options are available in every version of Visual Studio. The Code Analysis tab, for example, only appears in Visual Studio Premium and Ultimate. So don't panic if your screen looks a little different from this one. (You are opening these screens, right?)



# ADD AN ICON

All Windows applications need an icon. If you don't provide one, Visual Studio will use the default icon. The default image isn't very exciting, and it doesn't distinguish your application from all the others out there. So let's use the Project Properties dialog to add a custom icon to Hello, World. To make that happen in a WPF application (other application types can be a little different), we need just three steps:

- 1 Specify the icon file in the Application tab of Project Properties.
- 2 Build the application to make the icon available.
- 3 Set the icon property of the window in the WPF Designer.



This is the default icon. Pretty boring, huh?



We'll replace it with this one that looks a bit cooler and represents what our application actually does. This icon file is called `conversation.ico`, and it's included with the sample code. You can use this one, or any other icon file you like (try searching for `*.ico` in the Windows Explorer). Just copy it to the application folder for our sample app.



## MAKE A NOTE

Visual Studio includes simple editors for most resource types, including icons, from the Resources tab of Project Properties, or you can use a third-party tool. You can even open most third-party tools right inside Visual Studio by right-clicking the resource and choosing `Open With...`

And if you're not feeling particularly artistic, there are lots of icon sets available for free or fee on the Web that you can use. Just be sure to respect the artists' terms of use, or the karma gods will get you, even if copyright law doesn't.

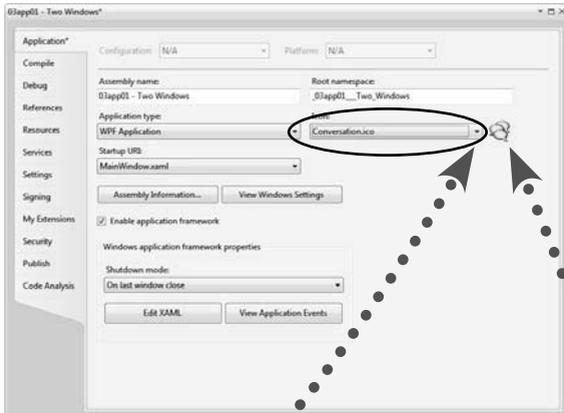
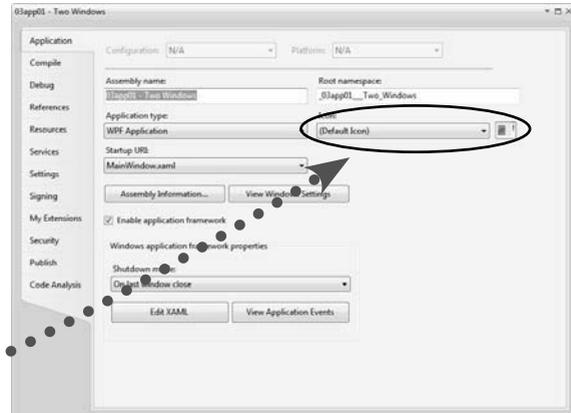


# SET THE APPLICATION ICON PROPERTY



If Hello, World isn't still open from the last exercise, open it from the Visual Studio Start Screen (it will be listed on the left side under Recent Projects) or from the File menu. Display the Project Properties using any of the techniques you've learned, and then select the Application tab.

We'll set the icon here.



Click here to open the dialog.

After you select your icon, Visual Studio will display a thumbnail of it here.



Open the combobox and select <Browse...>, and Visual Studio will display a standard File Open dialog. The first time you open the dialog, Visual Studio might take you to the Microsoft Visual Studio\Common7\IDE folder, which can be a little scary, but just navigate to the folder for the application and choose conversation.ico (or whichever icon file you chose to use).

After you select the icon and click the Open button, Visual Studio will show the icon on the tab.

# 2 BUILD THE APPLICATION

There are a lot of files involved in creating a Visual Studio application. In addition to the source files that you create, the resource files like icons that you create or reference, and the final executable created by the compiler, there are intermediate files that Visual Studio creates for you. When you set properties or resources, you need to tell Visual Studio to recreate some of these "behind-the-scenes" files so that they're available to other components like the designers. You do that by BUILDING the application.

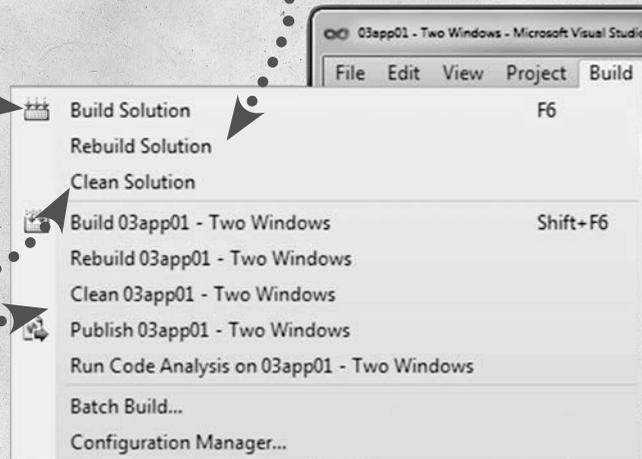
## THE BUILD MENU

Most of the time you can use Build Solution, which only rebuilds the files that have changed.

Clean Solution removes any intermediate files created by Visual Studio but doesn't build them. If Rebuild Solution seems to be acting strangely, try cleaning the solution first.

These options build and clean just the Project, not the whole Solution. Because our application only has a single project, there isn't any real difference, but when you have a lot of Projects in a Solution, these alternatives can save a lot of time.

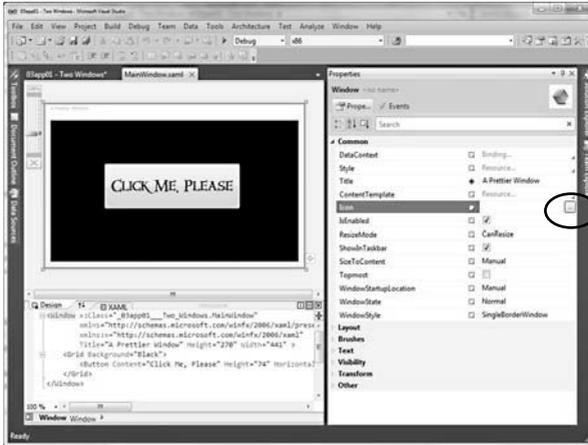
Rebuild Solution is a safer, but slower, choice. It rebuilds all the files, whether they've changed or not.



**A** Press F5 or choose Build Solution from the Build menu so that the icon will be available to the WPF Designer.

# 3

# SET THE WINDOW PROPERTY



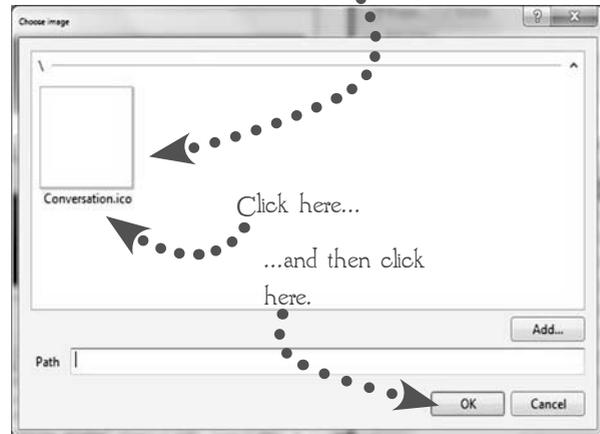
**A** If necessary, double-click `MainWindow.xaml` in the Solution Explorer to open the WPF Designer. In the Properties window, find the `Icon` property, select it, and then click the ellipsis button.

Click here.

**B** After you click the ellipsis in the Properties window, Visual Studio will display the Choose Image dialog. Your new icon will be displayed, but there might not be a thumbnail. It's okay; Visual Studio just hasn't caught up with us.

Click on your icon, and then click OK to set the property.

It's not a problem that there's no thumbnail.



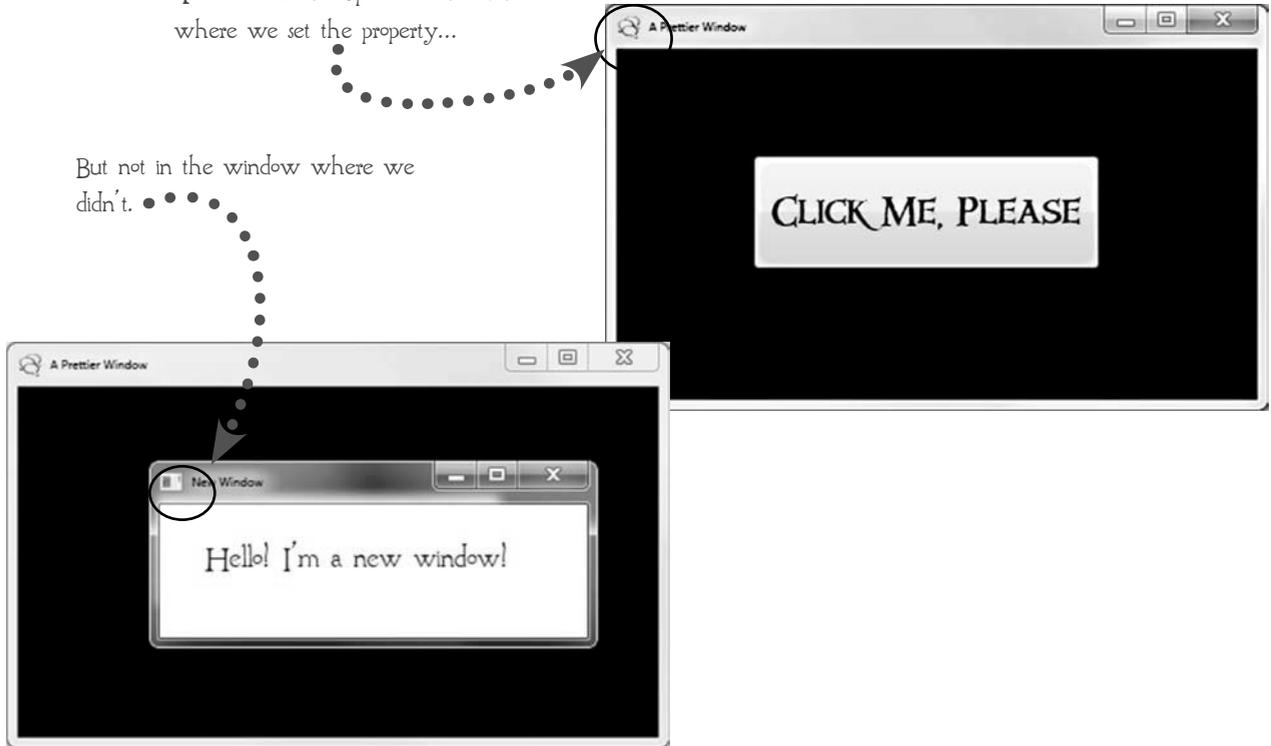
After you click the OK button, Visual Studio will set the property to a long value that begins with `pack://application.../`. That's just WPF-Speak for “look in the application file”, and we'll figure out how it all works later when we examine WPF Resources.

# DID IT WORK?

Don't take my word for it. Run the application and find out...

The icon shows up in the window where we set the property...

But not in the window where we didn't.



The application executable also uses the icon, as you can see in the bin/debug folder in Windows Explorer.



# ONE MORE TIME...

Let's run through the steps to add an application icon one more time...

1

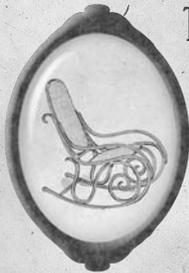
Add the icon file to the application in Project Properties.

2

Build the application to make the application available in the Designers.

3

Set the Icon property of the windows where you want the icon displayed.



## TAKE A BREAK

You've finished the first task of this chapter, so take a short break to let it all settle before moving on. But before you go, stop for a minute to think about what you've achieved...

- You created an application that displayed a Window and a MessageBox.
- You changed the appearance of the Window.
- You added a second window to the Project and wrote the code to display it.
- You added an icon to the application and the window.

That's a lot when you see it listed like that, isn't it? Go, you!



# REVIEW

Just a few exercises before we move on...

Solutions and Projects, Projects and Solutions. One's like a file folder, one's like a filing cabinet. Which is which?

Solution

Project

List three ways to add a Project:

①

②

③

List two ways to show the Solution Properties dialog:

①

②

List three ways to show Project Properties:

①

②

③

Change the application icon to something else. What happens to the window?

In the walkthrough, we only changed the icon of the main window. Add it to the other window in the application, as well.



# TAKE CONTROL

Visual Studio is a Windows application, and for the most part it behaves like any Windows application, with menus and toolbars and document windows where you do your work. But the work you do in Visual Studio is quite specialized, and the IDE adds some special capabilities to make it possible to work just the way you want to work.

This is a document window displayed as a tab, the default view.

You can display documents in tabs or as separate windows. You can even drag them outside the main IDE window.

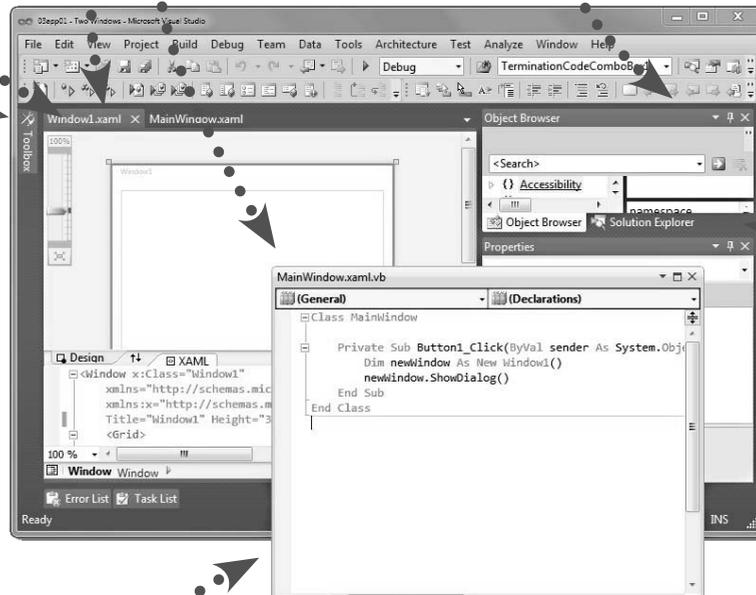
Tool windows can be docked to the edge of the IDE window or each other. When docked, they can be opened or closed.

This is a tool window that is docked and closed.

Tool windows that are docked to each other are displayed as tabs.

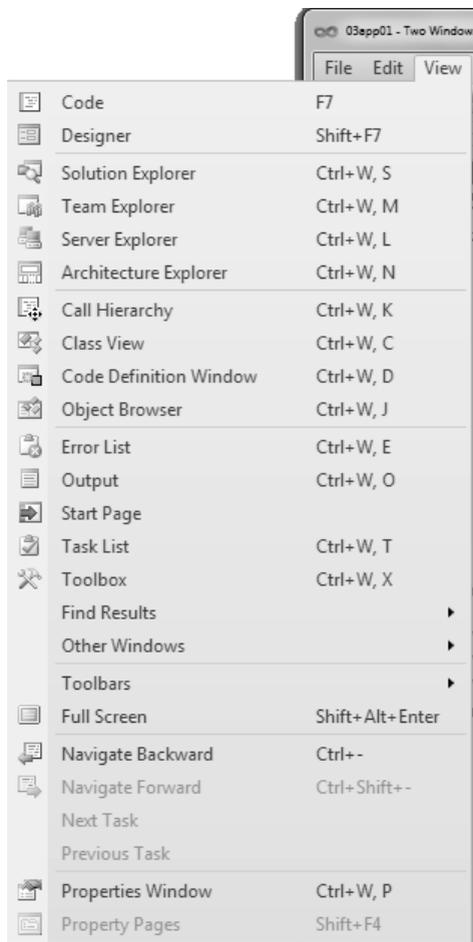
This is a document displayed in a floating window. It's not constrained by the Visual Studio IDE window.

This is a tool window that is docked and open.



# ARRANGING WINDOWS

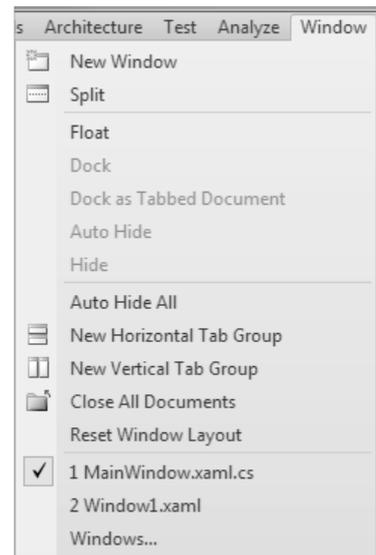
Like any Windows application, you can control the individual windows in Visual Studio through the View and Window menus or by dragging the title bar of a window.



The View menu controls the display of tool windows. (Don't confuse "tool window" with "Toolbox". The Toolbox is a tool window, but so are the Properties window and the Solution Explorer.)

Do you remember how to open a document window? Double-click on its name in the Solution Explorer.

The Window menu controls the display of open windows in the IDE. The most important item on this menu might be Reset Window Layout, which puts everything back in place when you get things messed up. (And you will, trust me.)



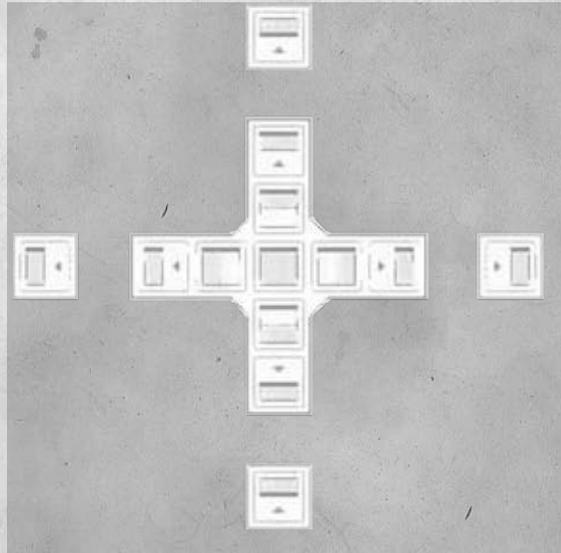


## ON YOUR OWN

The best way to learn how to control the windows in Visual Studio is to play with them, so take a few minutes to move the Object Browser around. The Object Browser lets you look through a class hierarchy. You'll find out what that means in Chapter 7.

Remember, you can always start over by choosing **Reset Window Layout** from the **Window** menu...

- 1 Show the Object Browser by selecting its name on the **View** menu. It will probably display as a tab. If it doesn't, drag its title bar until it does.
- 2 Make it float by dragging its title bar, or selecting **Float** from the **Window** menu.
- 3 Dock it to the left side of the screen along with the **Toolbox**. It will be open when you first dock it, so click the  to collapse it.
- 4 Drag it over so it displays as a tab in the same pane as the **Solution Explorer**.
- 5 Use **Reset Window Layout** to put everything back the way it was originally.
- 6 When you're dragging a window around, Visual Studio will display this odd docking widget. Can you work out what each bit does?



# MORE THAN EDITING...

Basic text editing in Visual Studio complies with Microsoft Windows standards. You can double-click to select a word, ctrl-click to extend the selection, and cut, copy or paste selections just the way you're used to. But the Visual Studio Code Editor also includes a seemingly magical tool called Intellisense that turns it from a stenographer into a personal assistant.

Intellisense will even create standard code and objects for you.

What colors does the new gizmo come in?

Intellisense shows you what an object can do as you type, so you don't need to memorize a lot of detailed syntax.

Send this guy a "Don't call us, we'll call you" letter.

What are the specifications for this new gizmo?



# LIST MEMBERS

The Intellisense capability that you'll probably use most often is the LIST MEMBERS function that displays a list of valid "things" that can be inserted where you're typing. You saw the list members function when you built your first application in Chapter 2.

As soon as you type a single character, the List Members box will open. Visual Studio is pretty smart about knowing what you can do, and it won't list things that don't apply (but it's not very smart about what makes sense).

To insert the selected item, press the Tab key or type the character after the item. In this case, that would be the "(" character, and you know that because of the definition displayed in the little help box on the far left.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    Dim newWindow As New Window1()
    newWindow.Show()
End Sub
```

Public Sub Show()  
Opens a window and returns without waiting for the newly opened window to close.

- Show
- ShowActivated
- ShowDialog
- ShowInTaskbar

Common All

When you select an item in the box, Visual Studio shows you the definition of the item, a short description, and, if the item is a method (something an object knows how to do), any exceptions the method can throw in a little help box.

To highlight an item in the List Members box, you can keep typing characters to limit the list or use the up and down arrow keys.

The box lists everything that contains the letters you type, not just the ones that begin with what you typed. Typing "n", for example, would display both "NewItem" and "EditNew", assuming they were available.

An EXCEPTION is the way an object lets the rest of the program know something is wrong. We'll look at exceptions and what to do about them in Chapter 7.

## ON YOUR OWN

Try using Intellisense to change the ShowDialog() method call in the sample application to Show(). Run it. How is the behavior different?

# PARAMETER INFORMATION

METHODS are things that objects can do. (We'll look at exactly what "object" and "method" mean in detail in Chapter 8.) Some methods take PARAMETERS, which are bits of information that you pass to the method to control exactly how it does whatever it is it does. In the bad old days before Intellisense, programmers spent a lot of time trying to remember exactly what parameters a method took, and in what order. The Intellisense Parameter Info box eliminates all that by showing you exactly what your options are.

As soon as you type the opening paren of a method call, Intellisense displays the Parameter Info box that shows you the method definition, a description of the method, and of the first parameter.

```
MessageBox.Show("A Message",|
```

▲ 1 of 5 ▼ Show(messageBoxText As String, caption As String) As System.Windows.MessageBoxResult  
Displays a message box that has a message and title bar caption; and that returns a result.  
**caption:** A System.String that specifies the title bar caption to display.

If a method has different versions, you can use the up and down arrow keys to scroll through them.

As you type each parameter, Intellisense updates the display to show the description of the next one.

```
MessageBox.Show("A Message", "caption",
```

▲ 1 of 4 ▼ Show(messageBoxText As String, caption As String, button As System.Windows.MessageBoxButton) As System.Windows.MessageBoxResult  
Displays a message box that has a message, title bar caption, and button; and that returns a result.  
**button:** A System.Windows.MessageBoxButton value that specifies which button or buttons to display.

System.Windows.MessageBoxButton.OK = 0  
The message box displays an OK button.

☐	MessageBoxButton.OK
☐	MessageBoxButton.OKCancel
☐	MessageBoxButton.YesNo
☐	MessageBoxButton.YesNoCancel
Common   All	

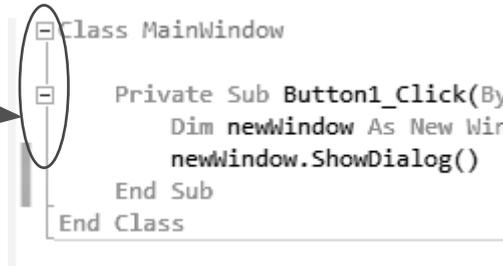
List Members works right alongside Parameter info, so sometimes the screen can get a bit crowded! Mostly it's helpful, but if all the windows get in your way, you can always make them go away by pressing ESC.

# ROAD MAPS

Several editing functions help you keep track of where you are and what you've done. Using them is pretty intuitive, but here's a quick rundown:

## CODE OUTLINING

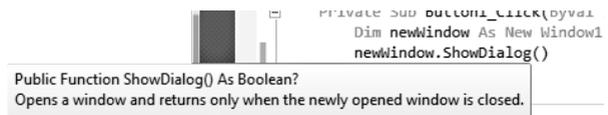
Click on the + or = characters in the left margin to collapse and expand units of code.

A code editor snippet showing a class structure. The left margin contains a vertical line with a minus sign (-) next to 'Class MainWindow' and a plus sign (+) next to 'Private Sub Button1\_Click'. A dotted arrow points from the text 'Click on the + or = characters...' to the plus sign in the margin. The code is as follows:

```
Class MainWindow
  Private Sub Button1_Click(By
    Dim newWindow As New Win
      newWindow.ShowDialog()
    End Sub
End Class
```

## QUICK INFO

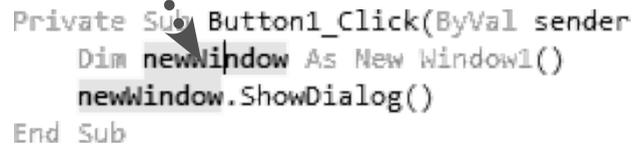
Hover the mouse over any identifier, and Intellisense will display its definition and a description. (We'll see how to create descriptions for the code you write a bit later.)

A screenshot of a code editor with a tooltip. The tooltip is a white box with a black border containing the text: 'Public Function ShowDialog() As Boolean? Opens a window and returns only when the newly opened window is closed.' The tooltip is positioned over the 'newWindow.ShowDialog()' call in the code below.

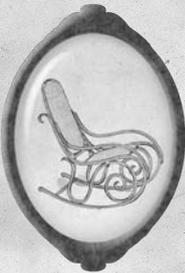
```
Private Sub Button1_Click(ByVal
  Dim newWindow As New Window1
    newWindow.ShowDialog()
```

## REFERENCE HIGHLIGHTING

Select any symbol in the editor by clicking in it, and Visual Studio will highlight all the references to that symbol in the code.

A code editor snippet showing a sub procedure. The code is as follows:

```
Private Sub Button1_Click(ByVal sender
  Dim newWindow As New Window1()
    newWindow.ShowDialog()
End Sub
```

A dotted arrow points from the text 'Select any symbol...' to the 'newWindow' variable in the code. The variable 'newWindow' and its call 'newWindow.ShowDialog()' are highlighted in grey in the original image.

## TAKE A BREAK

You've almost finished this chapter, so take a short break before you come back for the final Review.



# REVIEW

Name at least one way to add a Project to a Solution:

Name at least one way to add a File to a Project:

Where would you assign an application icon?

What does this widget do?



On a new, blank line inside the `Button1_click` event handler, type an “a” to trigger Intellisense. What’s the description of the Array object?

How many versions of the `Array.BinarySearch()` method are there?

Hover the mouse over one of the instances of the `Window` identifier. (There are two in your source file.) What’s the description of the window?

**Congratulations!** You've finished the chapter. Take a minute to think about what you've accomplished before you move on to the next one...

List three things you learned in this chapter:

①

②

③

Why do you think you need to know these things in order to be a C# programmer?

Is there anything in this chapter that you think you need to understand in more detail? If so, what are you going to do about that?



# INDEX

## A

- abstract classes, 404, 409-412
- abstract interfaces, 409-412
- abstract types, 403
- abstraction, 384, 391-392, 394
- access modifiers, 223
  - in Class Designer, 232-234
  - for get/set accessors, 246-249
  - in information hiding, 241
- accessors, 241-249
- actions, 25
- AddHandler keyword, 479
- ADO.NET, 486
- ADO.NET Data Services, 486
- ADO.NET Entity Framework, 486
- adorners, 769
- aggregate types. *See* arrays; generics; specialized sets
- Agile Development, 20
- Agile Manifesto, 20
- Alexander, Christopher, 457
- alignment of panel controls, 542
- angle brackets in syntax diagrams, 128
- animated values, 596
- animations, 747-755
  - class hierarchy, 748
  - example of, 752-755
  - reasons for using, 747
  - storyboards, 749-750
  - timeline classes, 751
- application development. *See also* OOA&D (Object-Oriented Analysis & Design); WPF controls
  - compilation process, 33-41
    - compiled versus interpreted languages, 34-35
    - JIT (just-in-time) compilation, 36-39
  - design patterns. *See* design patterns
  - “Hello, World” example, 53-58
  - steps in, 9-11, 13, 16-18, 59-63
  - system design, 19-32
    - Agile Development, 20
    - database schemas, 29
    - screen layouts, 29
    - UML class diagrams, 29
    - UML state diagrams, 25-28
    - use cases, 21-24
    - waterfall models, 20
- application facade, 487
- ApplicationException class, 210
- applications
  - adding buttons, 97-98
  - adding icons, 76-82
  - building, 78
  - deployment, 75, 107-119
    - adding shortcuts to setup programs, 114
    - build configurations, 116-117
    - ClickOnce, 108-109
    - creating setup programs, 110-112
    - setup designers, 113
    - types of, 107
  - running, 58
- architectural patterns, 459, 482-491
  - logical layers
    - business layer, 487
    - data access layer, 486
    - list of, 485
    - presentation layer, 491
    - service layer, 488-490
  - types of, 482-484
- ARGB color model, 674
- ArgumentException class, 210
- arguments, 173
  - reference types versus value types, 296-301
  - routed events, 635-644
- arranging windows, 84
- Array class, 360-362
- ArrayList class, 363-366
- arrays, 354-362
  - Array class, 360-362
  - creating, 355
  - For Each...Next statement, 359
  - initialization, 356
  - referencing elements in, 357-358
  - specialized sets and generics versus, 372
- AS <type> specifier, 128
- ASP.NET Web Forms, 491

- assemblies, 232
- assignment operator, 159
- attached properties, 538-541
- attribute syntax (XAML), 509
- attributes, 148

## B

- back end, 484
- backing fields, 241, 593
- bad code, characteristics of, 427
- BAML (Binary Application Markup Language), 523-527
- base value, determining, 597
- BasedOn property, 734-735
- behavior, 158, 171, 720
- Berra, Yogi, 425
- best practices, 13, 16
- binding expressions, 798
- binding source, 798, 807-810
- binding target, 798
- BindingMode property, 805
- bindings, 797
  - binding modes, 805
  - binding source, 807-810
  - to collections, 814-842
    - building bindable collections, 817-818
    - data templates, 819-820
    - data triggers, 821-822
    - filtering collections, 836-842
    - master-detail bindings, 823-824
    - object sharing, 830-832
    - panel templates, 819-820
    - sorting collections, 833-835
    - value conversions, 829
    - views, 825-828
  - creating, 802-804
  - DataContext property, 811
  - reasons for using, 799-800
  - structure of, 798
  - template bindings, 779-780
  - update triggers, 806
- bitmap effects, 712
- block elements, 708
- blurs, 713
- Boole, George, 165
- Boolean expressions, 165-169
- bounding boxes for gradients, 682
- boxing, 295
- Break All command (break mode), 104
- break mode, 101-106
- breakpoints, setting, 102
- Brookes, Frederick P., 20
- brushes, 677-697
  - creating explorer windows for, 678-680
  - gradients, 681-690
    - bounding boxes for, 682
    - linear gradients, 683-687
    - radial gradients, 688-690
  - tile brushes, 691-696
  - types of, 677
- bubbling events, 634
- bugs. *See also* exceptions
  - break mode, 101-106
  - Code Editor display of, 99-100
  - history of terminology, 59
  - in .NET Framework, 644
  - number in programs, 93
  - types of, 96
- build configurations, 116-117
- building applications, 78
- Burger, Andrej, 644
- business entities, 487
- business layer, 485, 487
- business rules, 487
- business workflows, 487
- Button class, 308
- buttons, 565-568
  - adding to applications, 97-98
  - class hierarchy, 565
  - content properties (XAML), 566
  - radio button groups, 567-568
- ByRef keyword, 301

## C

- Calendar class, 335
- calendars, historical date problems with, 338-339
- Call Stack window, 101
- callbacks, 594, 602, 614-623
  - coerce value callbacks, 616
  - property changed callbacks, 617
  - registering, 618-623
  - validation callbacks, 615
- calling methods, 175-176
- camel case, 132
- CanBe relationship, 386
- CanDo relationship, 288, 386
- Canvas control, 537

- casting, 139, 290
  - explicit casting, 292
  - implicit casting, 291
  - is keyword, 293-294
  - polymorphism and, 397
  - TryCast() function, 293-294
  - Type...Is operator, 398-400
- change tracking and notification with dependency
  - properties, 595
- chaotic cohesion, 432
- Char structure, 327-328
- character data, 327-334
  - Char structure, 327-328
  - comparing strings, 330
  - escaping text, 329
  - String class methods, 331-332
  - String versus StringBuilder class, 333
- CheckBox control, 565
- chronological data, 335-345
  - DateTime class, 337-340
  - DateTime versus DateTimeOffset classes, 336
  - DateTimeOffset class, 343-344
  - Timespan class, 341-342
- Class Designer, 227-238
  - abstract classes in, 410
  - access modifiers in, 232-234
  - enumerations, creating, 282
  - fields in, 235
  - inheritance in, 237
  - properties in, 231, 236
  - structures, creating, 276-278
- Class Details pane (Class Designer), 236
- class diagrams, 29, 225-238
- Class Name ComboBox (Code Editor), 640
- classes. *See also* FCL (Framework Class Library)
  - abstract classes, 404, 409-412
  - access modifiers in Class Designer, 232-234
  - components of, 224
  - control classes, 553-556
  - default implementations, extending, 415-416
  - definition syntax, 239-240
  - degenerate classes, 446
  - dependency properties, 602-603
  - design principles. *See* design principles
  - extension methods, 419-421
  - fields, 235, 241-249
  - inheritance, 210, 237
  - instances and constructors versus, 259
  - interfaces versus, 289
  - items controls class hierarchy, 570-571
  - methods, 250-267
    - constructor chaining, 260
    - creating, 252-258
    - me keyword, 256
    - syntax, 254
    - types of, 251
  - modules, 405-408
  - within .NET Framework, 221-223
  - OOB principles. *See* OOB principles
  - properties, 241-249
    - accessor access, 246-249
    - in Class Designer, 231, 236
    - implementation, 242-245
  - relationship with objects, 221
  - for routed commands, 663
  - sealed classes, 402-404, 417-418
  - sealed members, 404
  - static classes, 404
  - static font classes, 707
  - static members, 404
  - structures versus, 275
  - timeline classes, 751
  - type modifiers. *See* type modifiers
  - UML class diagrams, 29, 225
  - virtual members, 404
  - WPF class hierarchy, 535
- ClickOnce, 107-109
- client/server applications, 482
- CLR (Common Language Runtime), 36
- CLR-compliant languages, JIT (just-in-time)
  - compilation, 36-39
- CMYK color model, 674
- Code Complete* (McConnell), 184
- Code Editor, 62, 86-89, 640
- code outlining, 89
- code smells, characteristics of, 427
- CodePlex open source site, 669
- coerce value callbacks, 616
- cohesion, 430-437
- collection property syntax (XAML), 513-514
- collections, 353
  - binding to, 814-842
    - building bindable collections, 817-818
    - data templates, 819-820
    - data triggers, 821-822
    - filtering collections, 836-842

- master-detail bindings, 823-824
  - object sharing, 830-832
  - panel templates, 819-820
  - sorting collections, 833-835
  - value conversions, 829
  - views, 825-828
- XAML, 513-514
- CollectionView, 798
- color, 674-676
  - gradients, 681-690
    - bounding boxes for, 682
    - linear gradients, 683-687
    - radial gradients, 688-690
- color profiles, 675
- Color structure, 675
- ColorInterpolationMode property, 687
- ComboBox control, 575-576
- commands
  - in break mode, 104
  - control-of-flow commands, 184-204
    - iteration commands, 191-197, 359
    - jump commands, 198-204
    - selection commands, 186-190
    - types of, 185
  - exception handling commands, 184, 205-217
    - Exception class, 209-212
    - Try...Catch...Finally command, 208
  - routed commands, 629-630, 654-666
    - class hierarchy, 663
    - creating, 665
    - in FCL (Framework Command Library), 656-658
    - hooking up, 660-662
    - input gestures, 664-665
    - logical design, 655, 659
    - types of, 181, 183-184
- comments, 122-123, 141-147
  - line comments, 142-143
  - Task List comments, 144-145
  - XML comments, 146-147
- Common Language Runtime (CLR), 36
- communicational cohesion, 432
- comparing strings, 330
- compartments, 225
- compilation errors, 96-100
- compilation process, 33-41
  - compiled versus interpreted languages, 34-35
  - JIT (just-in-time) compilation, 36-39
  - for XAML, 523-527
- compiled languages, interpreted languages versus, 34-35
- compilers, 17
- Concat() method, 331
- concatenation operator, 159-160
- concurrency, 825
- conditional iteration, 192-194
- #const directive, 149
- constants, 127-128. *See also* declared elements
- constructors, 135, 251
  - base constructors, 416
  - chaining, 260
  - classes and instances versus, 259
  - default constructors, 258
- content controls, 533, 536, 557-569
  - buttons, 565-568
  - decorators versus, 773
  - headered content controls, 563-564
  - Window control, 558-562
- content properties (XAML), 510, 566
- continuation character (`_`), 124
- Continue command (break mode), 104
- Continue command (jump commands), 200-202
- control classes, 553-556
- control contracts, 784-787
- control templates, 721, 765, 775-787
  - control contracts, 784-787
  - example of, 776-778
  - necessary control components, 783
  - presenters, 781-782
  - syntax, 775
  - template bindings, 779-780
  - visual states, 788-794
  - visual transitions, 791-792
- control-of-flow commands, 181, 184-204
  - iteration commands, 191-197, 359
  - jump commands, 198-204
  - selection commands, 186-190
  - types of, 185
- controls. *See* WPF controls
- converters, 798
- converting
  - reference types to/from value types, 295
  - values in bindings, 829
- Cooper, Alan, 491
- counters, 195
- CreateInstance() method, 360
- currency, 825
- Custom Actions Editor, 113

- custom events, creating, 647
- Custom keyword, 647
- customizing
  - Visual Studio UI, 83-85
  - WPF controls. *See* control templates; graphics

## D

- data access layer, 485-486
- data binding, 495
- data contracts, 488
- Data Source window, 68
- data stores, 485
- data templates, 819-820
- Data Tips in break mode, 103
- data triggers, 736, 821-822
- data types. *See* types
- data validation. *See* validation
- database schemas, 29
- DataContext property, 811
- dataset projects, 70
- dates/times. *See also* chronological data
- DateTime class, 335-340
- DateTimeOffset class, 335-336, 343-344
- debug build configuration, 116
- DEBUG constant, 151
- Decimal type, 321, 324
- declarations, 122-123
  - of arrays, 355
  - of events, 646-647
  - of methods, 254
  - of namespaces, 515-516
  - syntax, 128-130
- declared elements, 123
  - components of, 134
  - declaring, 128-130
  - memory allocation, 133
  - New keyword, 135
  - Option Infer, 138-139
  - scope, 262-265
  - types of, 127
- decorators, 769, 773-774
- default constructors, 258
- default implementations, extending, 415-416
- default settings, Visual Basic language elements, 136-137
- degenerate classes, 446
- delegates, 615
- dependency inversion, 441-444, 463
- dependency properties, 591
- callbacks, 614-623
  - coerce value callbacks, 616
  - property changed callbacks, 617
  - registering, 618-623
  - validation callbacks, 615
- capabilities of, 595
- classes, 602-603
- creating, 604-609
- registering, 610-612
- reusing, 613
- routed events versus, 628, 645
- value calculation, 596-601

DependencyProperty class, 602-603

deployment, 75, 107-119

- adding shortcuts to setup programs, 114
- build configurations, 116-117
- ClickOnce, 108-109
- creating setup programs, 110-112
- setup designers, 113
- types of, 107

Dequeue() method, 367

design documents, 71

design patterns, 16, 457

- architectural patterns, 482-491
  - business layer, 487
  - data access layer, 486
  - logical layers, list of, 485
  - presentation layer, 491
  - service layer, 488-490
  - types of, 482-484
- Observer pattern, 469-481
  - event mapping, 631
  - handling events, 477-479
  - logical observer, 470-471
  - .NET solution, 475
  - raising events, 476
  - solution options, 472-474
- presentation patterns, 492-498
  - Model-View-Controller pattern, 495
  - Model-View-Presenter pattern, 496
  - Model-View-ViewModel pattern, 497
  - pros and cons, 493-494
  - reasons for using, 460-461
- Strategy pattern, 463-468

*Design Patterns: Elements of Reusable Object-Oriented Software* (Gamma et al.), 457

design principles

- characteristics of bad code, 427

- Law of Demeter, 449-453
- Liskov Substitution Principle, 445-448
- Open/Closed Principle, 438-444
- Single Responsibility Principle, 431-437
- when to violate, 428-429
- designer files, 71
- design-time, 34
- destructors, 251
- development environments, 11
- development methodologies, 13, 16
  - Agile Development, 20
  - waterfall models, 20
- device independent pixel, 540
- diagrams. *See* UML (Unified Modeling Language)
- Dictionary class, 367
- Dim keyword, 128
- direct events, 630, 634
- directives, 122-123, 148-151
- DirectX, 712
- “Discover a Series of Fortunate Event Handlers in Visual Basic” (Getz), 647
- DivideByZeroException class, 210
- dll files, 35
- DockPanel control, 537, 544-546
- document outline window, 68
- documents, flow, 708-711
- Do...Loop command, 192-194
- dot operator, 172
- Double type, 321, 323
- downloading
  - source code, 6
  - Visual Studio, 6
- drawing shapes, 770-772
- drop shadows, 713
- dynamic event handlers, 479
- dynamic resources, 727-728

**E**

- Edison, Thomas, 59
- effects (graphics), 712-714
- Einstein, Albert, 425
- embedding fonts, 707
- empty bindings, 814
- encapsulation, 384, 391, 393
- encoding, 327
- ENIAC, 59
- Enqueue() method, 367
- enum keyword, 284
- enumerations, 269, 274, 281-287
  - creating in Class Designer, 282
  - methods in, 284-287
  - within .NET Framework, 270-271
  - syntax, 283
  - validation, 283
- equations, expressions versus, 160-162
- Error List window, 100
- errors. *See* bugs
- escaping text, 329
- event handlers, 57
  - adding to buttons, 98
  - for routed events, 639-641
  - event triggers, 736, 744-746
- events, 25, 224
  - adding to windows, 57
  - bubbling events, 634
  - custom events, creating, 647
  - declaring, 646-647
  - direct events, 630, 634
  - handling, 477-479
  - mapping to Observer pattern, 631
  - raising, 476
  - routed events, 627, 632-653
    - arguments, 635-644
    - creating, 648-653
    - dependency properties versus, 628, 645
    - reasons for using, 632-633
    - strategies, 634
  - tunneling events, 634
- Exception class, 209-212
- exception handling commands, 181, 184, 205-217
  - Exception class, 209-212
  - Try...Catch...Finally command, 208
- exceptions, 87, 96. *See also* bugs
- exe files, 35
- executables, creating, 33-41
  - compiled versus interpreted languages, 34-35
  - JIT (just-in-time) compilation, 36-39
- Exit command, 199
- Expander control, 563
- explicit casting, 292
- explicit iteration, 195-196
- explorer windows, creating, 678-680
- expressions, 98, 155
  - literal expressions, 158-170
  - Boolean expressions, 165-169
  - equations versus, 160-162

- operator precedence, 163-164
  - syntax, 159
- object expressions, 158, 171-178
  - calling methods, 175-176
  - member access, 172
  - method signatures, 173
  - overloading methods, 174
  - type names, 177
- types of, 157-158
- extending default implementations, 415-416
- extension methods, 407, 419-421

## F

- FCL (Framework Class Library)
  - arrays, 354-362
    - Array class, 360-362
    - creating, 355
    - For Each...Next statement, 359
    - initialization, 356
    - referencing elements in, 357-358
  - character data, 327-334
    - Char structure, 327-328
    - comparing strings, 330
    - escaping text, 329
    - String class methods, 331-332
    - String versus StringBuilder class, 333
  - chronological data, 335-345
    - Date Time class, 337-340
    - Date Time versus DateTimeOffset classes, 336
    - Date Time Offset class, 343-344
    - Timespan class, 341-342
  - functionality example, 308
  - generics, 372-377
  - namespaces, 310-316
    - creating, 315
    - Imports statement, 312-314
  - numeric data, 321-326
    - Decimal type, 324
    - floating point numbers, 323
    - integer efficiency tips, 322
    - Math class, 325
  - Object Browser, 317-320
  - organization, 306-307, 350-351
  - specialized sets, 363-371
    - ArrayList class, 363-366
    - LIFO, FIFO, linked lists, key/value pairs, 367-371
- FCL (Framework Command Library), routed commands in, 656-658

- fields, 224, 241-249
  - in Class Designer, 235
  - dependency properties and, 592
  - naming conventions, 236
- FIFO (first in, first out), 367-371
- File System Designer, 113
- File Type Editor, 113
- files, adding to projects, 70-73
- filtering collections, 836-842
- fixed documents, 708
- floating point numbers
  - integers versus, 139
  - precision, 323
- flow control. *See* control-of-flow commands
- flow controls, 711
- flow documents, 708-711
- FlowDocumentPageViewer control, 711
- FlowDocumentReader control, 711
- fonts, 704
  - embedding, 707
  - static font classes, 707
- For Each...Next statement, 359
- For...Next command, 195-196
- fragile code, 427
- Framework Class Library. *See* FCL (Framework Class Library)
- frameworks. *See* .NET Framework
- Fraser, Bruce, 676
- front end, 484
- functional cohesion, 432
- functions, 173

## G

- gamut, 675
- garbage collection, 265
- generics, 353, 372-377
  - arrays and specialized sets versus, 372
  - instantiation, 373
- gestures, input, 664-665
- get accessors, 241-249
- Getz, Ken, 647
- global variables, 407
- glyph runs, 691
- glyphs, 704
- GMT (Greenwich Mean Time), 336
- GoTo command, 203-204
- gradients, 681-690
  - bounding boxes for, 682

- linear gradients, 683-687
- radial gradients, 688-690
- graphics, 669-671
  - brushes, 677-697
    - creating explorer windows for, 678-680
    - gradients, 681-690
    - tile brushes, 691-696
    - types of, 677
  - color, 674-676
  - effects, 712-714
  - flow documents, 708-711
  - Mindbloom website versus Microsoft Word, 672
  - pens, 698-702
  - typography, 703-707
    - fonts and typefaces, 704
    - static font classes, 707
    - text characteristics, 705
    - text controls, 706
- Greenwich Mean Time (GMT), 336
- Gregorian calendar, 339
- Grid class, 308
- Grid control, 537, 547-548
- GridSplitter control, 549-550
- GroupBox control, 563
- grouping collections, 836-842

## I

- hacks, 144
- Handles keyword, 477, 641
- handling events, 477-479
- Harvard Mark II, 59
- HasA relationship, 288, 386
- Hashtable class, 367
- headered content controls, 563-564
- heap, 272-273
- height of panel controls, 543
- “Hello, World” example, 53-58
- High Level Shading Language (HLSL), 712
- historical dates, 338-339
- HLSL (High Level Shading Language), 712
- horizontal alignment of panel controls, 542
- HSV color model, 674
- Hungarian notation, 132

## I

- ICollection interface, 364
- icons, adding to applications, 76-82
- IDE (integrated development environment), 17

- ideas, 10
- identifiers, 128
  - as expressions, 157
  - naming conventions, 131-132
- IEnumerable interface, 364
- if statement, 186-188
- #if...#else...#endif directive, 149
- IList interface, 364
- Immediate window, 101
- immobile code, 427
- immutability, 333
- immutable properties, 247
- implementation of properties, 242-245
- implicit casting, 138-139, 291
- implicit styles, 731
- Imports statement, 312-314
- incompatible contracts, 447
- indexes, 354
- information hiding, 241-249, 393
- inheritance, 225, 384, 391-392
  - in Class Designer, 237
  - in Exception class, 210
- inheritance casts, 291
- initialization, 73
  - arrays, 356
  - of decimal values, 324
  - with New keyword, 135
- inline elements, 708
- input gestures, 664-665
- instances, 221, 251, 259
- instantiation, 221
  - generics, 373
  - structures, 279
- Int32 class, 308
- Int32 type, 321
- integers
  - efficiency tips, 322
  - floating point numbers versus, 139
- integrated development environment (IDE), 17
- Intellisense, 57, 62, 86-88, 146-147
- IntelliTrace commands (break mode), 104
- interfaces, 269, 274, 288-289
  - abstract interfaces, 409-412
  - for ArrayList class, 364
  - classes versus, 289
  - within .NET Framework, 270-271
  - programming to interfaces, 442, 463
  - relationships and, 288

- service interfaces, 488
- internal access modifier, 232
- interpreted languages, compiled languages versus, 34-35
- invoking methods, 175-176
- is keyword, 293-294
- IsA relationship, 288, 386
- items controls, 533, 536, 570-576
  - binding to collections, 814-842
    - building bindable collections, 817-818
    - data templates, 819-820
    - data triggers, 821-822
    - filtering collections, 836-842
    - master-detail bindings, 823-824
    - object sharing, 830-832
    - panel templates, 819-820
    - sorting collections, 833-835
    - value conversions, 829
    - views, 825-828
  - class hierarchy, 570-571
  - ComboBox control, 575-576
  - ListBox control, 575-576
  - TreeView control, 572-574
- iteration commands, 183, 185, 191-197
  - conditional iteration, 192-194
  - For Each...Next statement, 359
  - explicit iteration, 195-196

## J

- jagged arrays, 354
- JIT (just-in-time) compilation, 36-39
- Julian calendar, 339
- jump commands, 183, 185, 198-204
  - Continue command, 200-202
  - Exit command, 199
  - GoTo command, 203-204

## K

- key frame animations, 751
- keyed styles, 730
- key/value pairs, 367-371
- keywords, 131, 223
- kludges, 144

## L

- lambdas, 157
- Launch Conditions Editor, 113
- Law of Demeter, 430, 449-453
- layered architecture. *See* logical layers

- layout transforms, 758
- LIFO (last in, first out), 367-371
- line comments, 142-143
- linear gradients, 681-687
- lines, drawing, 698-702
- linked lists, 367-371
- linkers, 17
- linking, 35
- LINQ expressions, 157
- Lippert, Eric, 418
- Liskov Substitution Principle, 430, 445-448
- list members, 87
- ListBox control, 575-576
- ListView class, 571
- literal expressions, 158-170
  - Boolean expressions, 165-169
  - equations versus, 160-162
    - operator precedence, 163-164
  - syntax, 159
- local values, 597
- Locals window, 101
- logical errors, 96
- logical layers
  - business layer, 487
  - data access layer, 486
  - list of, 485
  - presentation layer, 491
  - service layer, 488-490
- logical operators, 166
- logical tree, 506, 519, 766
- loops. *See* iteration commands
- LSP. *See* Liskov Substitution Principle

## M

- margin of panel controls, 542-543
- master-detail bindings, 823-824
- Math class, 325
- MatrixTransform class, 756
- Max() method, 325
- McConnell, Steve, 184
- me keyword, 256
- measurement units, device independent pixel, 540
- member access, 172
- memory allocation, 133
  - garbage collection, 265
  - reference and value types, 272-273
  - scope, 262-265
  - in structures versus classes, 275

- meta types. *See* character data; chronological data; numeric data
- method calls, 175
- Method Name ComboBox (Code Editor), 640
- methods, 73, 88, 224, 250-267
  - for Array class, 360
  - calling, 175-176
  - cohesion levels, 432
  - constructor chaining, 260
  - constructors, classes and instances versus, 259
  - creating, 252-258
  - for Decimal type, 324
  - in enumerations, 284-287
  - extension methods, 407, 419-421
  - in Math class, 325
  - me keyword, 256
  - overloading, 174
  - passing by reference/value, 299-301
  - signatures, 173
  - static methods, 176, 479
  - in String class, 331-332
  - syntax, 254
  - types of, 251
- Microsoft Intermediate Language (MSIL), 36
- Microsoft Word, Mindbloom website versus, 672
- Min() method, 325
- Mindbloom website, Microsoft Word versus, 672
- Model-View-Controller pattern, 495
- Model-View-Presenter pattern, 496
- Model-View-ViewModel pattern, 459, 497
- modifiers. *See* access modifiers; type modifiers
- modules, 402, 405-408
- modulus operator, 161
- monolithic applications, 482
- MSIL (Microsoft Intermediate Language), 36
- multi-dimensional arrays, 354
- multi-line statements, 124
- multiple inheritance, 392
- multiple transformations, 758-759
- multiple trigger conditions, 739-740
- multi-statement lines, 124
- MustInherit keyword, 403, 410
- MustOverride keyword, 403, 410
- mutability, 333
- MVC (Model-View-Controller) pattern, 495
- MVP (Model-View-Presenter) pattern, 496
- MVVM (Model-View-ViewModel) pattern, 497
- MyBase keyword, 415-416

*The Mythical Man Month* (Brookes), 20

## N

- namespaces, 310-316
  - creating, 315
  - Imports statement, 312-314
  - XAML, 515-516
- naming conventions
  - abstract classes, 409
  - fields and properties, 236
  - identifiers, 131-132
  - variables, 264
  - Visual Basic versus .NET type names, 177
- Nash, John, 144
- negation operator, 160
- nesting panel controls, 551
- .NET Framework, 43
  - advantages of, 50-52
  - bugs in, 644
  - Class Library organization, 306-307, 350-351
  - classes within, 222-223
  - enumerations within, 270-271
  - interfaces within, 270-271
  - structures within, 270-271
  - type names, 177
  - types. *See* types
- .NET Platform, 43-65
  - components of, 44-49
  - “Hello, World” example, 53-58
- New keyword, 135
- Norman, Donald, 491
- NotImplementedException class, 210
- NotOverridable keyword, 402
- NotOverrideable keyword, 417-418
- n-tiered applications, 482
- numeric data, 321-326
  - Decimal type, 324
  - floating point numbers, 323
  - integer efficiency tips, 322
  - Math class, 325

## O

- Object Browser, 85, 317-320
- object code, 34
- object composition, 386
- object elements (XAML), 509
- object expressions, 158, 171-178
  - calling methods, 175-176

- member access, 172
  - method signatures, 173
  - overloading methods, 174
  - type names, 177
  - object hierarchy, 311
  - object sharing, 830-832
  - object tree, 506, 519
  - object-oriented programming principles. *See* OOP principles
  - objects
    - components of, 171
    - referencing from XAML, 831
    - relationship with classes, 221
    - state and behavior, 720
  - Observer pattern, 459, 469-481
    - event mapping, 631
    - handling events, 477-479
    - logical observer, 470-471
    - .NET solution, 475
    - raising events, 476
    - solution options, 472-474
  - OneTime binding, 805
  - OneWay binding, 805
  - OneWayToSource binding, 805
  - OOA&D (Object-Oriented Analysis & Design), 381-383
    - design principles
      - characteristics of bad code, 427
      - Law of Demeter, 449-453
      - Liskov Substitution Principle, 445-448
      - Open/Closed Principle, 438-444
      - Single Responsibility Principle, 431-437
      - when to violate, 428-429
    - OOP principles, 391-401
      - polymorphism and casting, 397
      - Type...Is operator, 398-400
    - type modifiers, 402-419
      - abstract classes and interfaces, 409-412
      - extension methods, 419-421
      - list of, 402-404
      - modules, 405-408
      - MyBase keyword, 415-416
      - sealed classes, 417-418
      - semi-abstract classes, 413
      - Shadows and Overrides keywords, 413-414
    - type relationships, 385-390
  - OOP principles, 391-401
    - polymorphism and casting, 397
    - Type...Is operator, 398-400
  - opaque code, 427
  - Open/Closed Principle, 430, 438-444
  - opening Project Designer, 75
  - operands, 159
  - operators, 159
    - dot operator, 172
    - logical operators, 166
    - precedence, 163-164
    - relational operators, 165
    - symbolic operators, 175
  - Option Compare, 137
  - Option Explicit, 137
  - Option Infer, 137-139
  - Option Strict, 137
  - ordinal string operations, 330
  - OverflowException class, 210
  - overloading methods, 174
  - overloads, 174, 257
  - Overridable keyword, 403
  - Overrides keyword, 413-414
- ## P
- padding of panel controls, 542-543
  - panel controls, 533, 536-552
    - attached properties, 538-541
    - decorators versus, 773
    - DockPanel control, 544-546
    - Grid control, 547-548
    - GridSplitter control, 549-550
    - nesting, 551
    - positioning and sizing, 542-543
  - panel templates, 819-820
  - paragraphs, 124
  - parameters, 88, 173, 253
  - Pascal case, 132
  - passing by reference, 299-301
  - passing by value, 299-301
  - A Pattern Language* (Alexander), 457
  - patterns. *See* design patterns
  - pens, 698-702
  - pixel shaders, 671
  - POCO (plain old CLR object), 592, 630
  - polymorphism, 384, 391, 393
    - casting and, 397
    - Type...Is operator, 398-400
  - Pop() method, 367
  - positioning panel controls, 542-543
  - precedence of operators, 163-164

- precision, floating point numbers, 323
- predicates, 360, 837
- preprocessor directives, 148. *See also* directives
- preprocessors, 148
- presentation layer, 485, 491. *See also* presentation patterns
- presentation patterns, 492-498
  - Model-View-Controller pattern, 495
  - Model-View-Presenter pattern, 496
  - Model-View-ViewModel pattern, 497
  - pros and cons, 493-494
- presenters, 767, 781-782
- primary colors, 676
- primary UI projects, 70
- Principle of Least Knowledge, 430, 449-453
- private access modifier, 232
- processing pipeline, 596
- programming to interfaces, 442, 463
- Project Designer, opening, 75
- Project Properties designer, 113
- projects. *See also* application development
  - adding files to, 70-73
  - adding to solutions, 70-73
  - creating, 54, 60
  - properties, 74-82
  - startup project, specifying, 74
  - steps in, 59-63
- properties, 224, 241-249
  - accessor access, 246-249
  - attached properties, 538-541
  - in Class Designer, 231, 236
  - content properties (XAML), 510, 566
  - dependency properties, 591
    - callbacks, 614-623
    - capabilities of, 595
    - classes, 602-603
    - creating, 604-609
    - registering, 610-612
  - reusing, 613
    - value calculation, 596-601
  - implementation, 242-245
  - for linear gradients, 687
  - naming conventions, 236
  - of solutions and projects, 74-82
- Properties window, 56-57, 68
- property changed callbacks, 617
- property element syntax (XAML), 509
- property triggers, 736-743
  - combining with styles, 741-742
  - multiple trigger conditions, 739-740
- PropertyMetadata class, 602-603
- protected access modifier, 232
- protected internal access modifier, 232
- public access modifier, 232
- publishing. *See* deployment
- Push() method, 367

## Q

- Queue class, 367

## R

- radial gradients, 681, 688-690
- radio button groups, 567-568
- RadioButton class, 308
- RadioButton control, 565
- raising events, 476
- range controls, 533
- read-only properties, 247
- Real World Color Management* (Fraser), 676
- redundant code, 427
- reference highlighting, 89
- reference types, 272-273, 275
  - converting to/from value types, 295
  - value types versus, 296-301
- referencing
  - array elements, 357-358
  - objects from XAML, 831
  - resources, 727-728, 830
  - XAML elements, 830
- reflection, 290
- registering
  - callbacks, 618-623
  - dependency properties, 610-612
- Registry Editor, 113
- relational operators, 165
- relationships, 288, 385-390
- RelativeSource class, 809-810
- release build configuration, 116
- RemoveHandler keyword, 479
- render transforms, 759
- resources, 71, 526, 719, 721
  - advantages and disadvantages of, 722
  - in code, 729
  - defining, 724-726
  - referencing, 727-728, 830
- REST (representational state transfer), 488
- Restart command (break mode), 104

- return types, 173
- reusing dependency properties, 613
- RGB color model, 674
- RichTextBox control, 711
- rigid code, 427
- root element (XAML), 515
- RotateTransform class, 756
- Round() method, 324-325
- routed commands, 629-630, 654-666
  - class hierarchy, 663
  - creating, 665
  - in FCL (Framework Command Library), 656-658
  - hooking up, 660-662
  - input gestures, 664-665
  - logical design, 655, 659
- routed events, 627, 632-653
  - arguments, 635-644
  - creating, 648-653
  - dependency properties versus, 628, 645
  - reasons for using, 632-633
  - strategies, 634
- running applications, 58
- runtime, 34
- runtime environments, 10, 17

**S**

- safe casts, 291
- scope, 262-265
- screen layouts, 29
- ScRGB color space, 687
- sealed classes, 402, 404, 417-421
- sealed members, 404
- select...case statement, 189-190
- selection commands, 183, 185-190
  - if statement, 186-188
  - select...case statement, 189-190
- Selector class, 571
- Sells, Chris, 418
- separation of responsibilities, 493
- sequential cohesion, 432
- servers, 484
- service interfaces, 488
- service layer, 485, 488-490
- service-oriented architecture, 484
- services, 484
- set accessors, 241-249
- sets. *See* arrays; generics; specialized sets
- setters, 597
- setup designers, 113
- setup programs, 107
  - adding shortcuts to, 114
  - build configurations, 116-117
  - creating, 110-112
- Shadows keyword, 413-414
- shapes, 769-772
- shared keyword, 402
- sharing objects, 830-832
- shortcuts, adding to setup programs, 114
- Show Next Statement command (break mode), 104
- side effects, 166
- signatures of methods, 173
- Silverlight, 491, 503
- Simonyi, Charles, 132
- simple statements, 124
- Single Responsibility Principle, 430-437
- Single type, 323
- single-dimensional arrays, 354
- singleton pattern, 407
- sizing panel controls, 542-543
- SkewTransform class, 757
- Snepscheut, Jan van de, 425
- SOA (service-oriented architecture), 488
- SOAP (simple object access protocol), 488
- Solution Explorer, 61, 68
- solutions, 67
  - adding projects to, 70-73
  - properties, 74-82
- sorting collections, 833-835
- source code, 11, 17
  - Code Editor, 62, 86-89
  - compiled versus interpreted languages, 34-35
  - downloading, 6
  - JIT (just-in-time) compilation, 36-39
  - steps in writing, 59-63
- source files, 71
- sparse storage with dependency properties, 595
- specialized sets, 363-371
  - ArrayList class, 363-366
  - arrays and generics versus, 372
  - LIFO, FIFO, linked lists, key/value pairs, 367-371
- specifications, 10, 16, 19-32
  - Agile Development, 20
  - database schemas, 29
  - screen layouts, 29
  - UML class diagrams, 29
  - UML state diagrams, 25-28

- use cases, 21-24
  - waterfall models, 20
- SpreadMethod property, 687
- square brackets in syntax diagrams, 128
- sRGB color space, 687
- stack, 272-273, 367
- StackPanel control, 537
- startup project, specifying, 74
- state, 158, 171, 720
- state diagrams, 25-28
- state members. *See* fields
- statements, 17, 122-123
  - class definitions, 239-240
  - commands. *See* commands
  - declared elements. *See* declared elements
  - expressions. *See* expressions
  - syntax, 124-126
- states, 25
- static classes, 404. *See also* modules
- static event handlers, 477
- static font classes, 707
- static members, 402, 404
- static methods, 176, 251, 479
- static resources, 727-728
- Step Into command (break mode), 104
- Step Out command (break mode), 104
- Step Over command (break mode), 104
- Stop Debugging command (break mode), 104
- stops, 681
- storyboards, 749-750
- Strategy pattern, 459, 463-468
- String class, 308
  - comparing strings, 330
  - escaping text, 329
  - methods, 331-332
- StringBuilder class versus, 333
- StringBuilder class, String class versus, 333
- StringComparison enumeration, 330
- strong typing, 139
- structured exception handling, 207-217
- structures, 269, 274-280
  - classes versus, 275
  - creating in Class Designer, 276-278
  - instantiation, 279
  - within .NET Framework, 270-271
- styles, 597, 721, 730-735
  - combining with triggers, 741-742
  - hierarchies in, 734-735

- types of, 730-731
- subs, 173
- symbolic operators, 175
- syntactic sugar, 135
- syntax errors, 96, 99-100
- system architectures, 13
- system design, 19-32
  - Agile Development, 20
  - database schemas, 29
  - screen layouts, 29
  - UML class diagrams, 29
  - UML state diagrams, 25-28
  - use cases, 21-24
  - waterfall models, 20
- SystemException class, 210

## T

- tab controls, 577
- targeted styles, 730
- Task List comments, 144-145
- template bindings, 797
- templates, 597
  - control templates, 721, 765, 775-787
    - control contracts, 784-787
    - example of, 776-778
    - necessary control components, 783
    - presenters, 781-782
    - syntax, 775
    - template bindings, 779-780
    - visual states, 788-794
    - visual transitions, 791-792
  - data templates, 819-820
  - panel templates, 819-820
- temporal cohesion, 432
- test projects, 70
- test-driven development, 70
- text. *See* character data; typography
- text editors, 17
- TextBox class, 308
- TextDecorations collection, 700-702
- themes, 597
- ticks, 337
- tile brushes, 691-696
- The Timeless Way of Building* (Alexander), 457
- timeline classes, 751
- times/dates. *See also* chronological data
- Timespan class, 335, 341-342
- TimeZoneInfo class, 335

- ToggleButton control, 565
- tokens, 144-145
- Toolbox, 68
- TRACE constant, 151
- transform groups, 758
- transformations, 671, 756-761
- transitions, 25, 791-792
- TranslateTransform class, 757
- TreeView control, 572-574
- triggers, 597, 721, 736-746
  - data triggers, 821-822
  - event triggers, 744-746
  - property triggers, 736-743
    - combining with styles, 741-742
    - multiple trigger conditions, 739-740
  - types of, 736
  - update triggers, 806
- true keyword, 160
- Truncate() method, 324-325
- TryCast() function, 293-294
- Try...Catch...Finally command, 208
- tunneling events, 634
- TwoWay binding, 805
- type keywords, 223
- type modifiers, 402-419
  - abstract classes and interfaces, 409-412
  - extension methods, 419-421
  - list of, 402-404
  - modules, 405-408
  - MyBase keyword, 415-416
  - sealed classes, 417-418
  - Shadows and Overrides keywords, 413-414
- type relationships. *See* relationships
- typefaces, 704
- Type...Is operator, 398-400
- types, 158, 171, 290-302. *See also* classes; enumerations; FCL (Framework Class Library); interfaces; structures
  - arrays, 354-362
    - Array class, 360-362
    - creating, 355
    - For Each...Next statement, 359
    - initialization, 356
    - referencing elements in, 357-358
  - boxing and unboxing, 295
  - casting, 290
    - explicit casting, 292
    - implicit casting, 139, 291
    - is keyword, 293-294

- polymorphism and, 397
  - TryCast() function, 293-294
  - Type...Is operator, 398-400
- character data, 327-334
  - Char structure, 327-328
  - comparing strings, 330
  - escaping text, 329
  - String class methods, 331-332
  - String versus StringBuilder class, 333
- chronological data, 335-345
  - DateTime class, 337-340
  - DateTime versus DateTimeOffset classes, 336
  - DateTimeOffset class, 343-344
  - Timespan class, 341-342
- generics, 372-377
- namespaces, 310-316
  - creating, 315
  - Imports statement, 312-314
- numeric data, 321-326
  - Decimal type, 324
  - floating point numbers, 323
  - integer efficiency tips, 322
  - Math class, 325
- reference and value types, 272-273, 296-301
- reflection, 290
- relationship with classes, 222-223
- specialized sets, 363-371
  - ArrayList class, 363-366
  - LIFO, FIFO, linked lists, key/value pairs, 367-371
  - Visual Basic versus .NET type names, 177
- typography, 703-707
  - fonts and typefaces, 704
  - static font classes, 707
  - text characteristics, 705
  - text controls, 706

## U

- UI (user interface) in Visual Studio, 61, 67-91. *See also*
  - graphics; WPF controls
  - adding solutions and projects, 70-73
  - Code Editor, 86-89
  - customizing, 83-85
  - parts of, 68
  - solution and project properties, 74-82
- UML (Unified Modeling Language)
  - UML class diagrams, 29, 225, 227-238
  - UML state diagrams, 25-28
- unboxing, 295

Unicode, 327-328  
Unified Modeling Language. *See* UML (Unified Modeling Language)  
units of measurement, device independent pixel, 540  
universal resource identifier (URI), 526  
Until keyword, 192-194  
update triggers, 806  
URI (universal resource identifier), 526  
use cases, 20-28  
user interface. *See* UI (user interface) in Visual Studio  
User Interface Editor, 113  
UTC (Universal Time, Coordinated), 336  
utility classes, 70

## V

validation  
    callbacks, 615  
    enumerations, 283  
value calculation with dependency properties, 595-601  
value conversions, 829  
value types, 272-273, 275  
    converting to/from reference types, 295  
    reference types versus, 296-301  
variables, 73, 98, 127. *See also* declared elements  
    garbage collection, 265  
    global variables, 407  
    implicit typing, 138-139  
    naming conventions, 264  
    passing by reference/value, 299-301  
    scope, 262-265  
vertical alignment of panel controls, 542  
Viewbox instance, 692-693  
Viewport instance, 692-693  
views, 825-828  
virtual keyword, 413  
virtual members, 403-404  
visibility, 232  
Visual Basic language elements  
    commands  
        control-of-flow commands, 184-204  
        exception handling commands, 184  
        types of, 181, 183-184  
    comments, 141-147  
        line comments, 142-143  
        Task List comments, 144-145  
        XML comments, 146-147  
    declared elements  
        components of, 134  
        declaring, 128-130  
        memory allocation, 133  
        New keyword, 135  
        Option Infer, 138-139  
        types of, 127  
    default settings, 136-137  
    directives, 148-151  
    expressions, 155  
        literal expressions, 158-170  
        object expressions, 158, 171-178  
        types of, 157-158  
    identifiers, naming conventions, 131-132  
    list of, 122, 156, 182  
    statements  
        class definitions, 239-240  
        syntax, 124-126  
        type names, 177  
Visual Basic, XAML versus, 507-508  
visual states, 788-794  
Visual Studio  
    Class Designer, 227-238  
        abstract classes in, 410  
        access modifiers in, 232-234  
        enumerations, creating, 282  
        fields in, 235  
        inheritance in, 237  
        properties in, 231, 236  
        structures, creating, 276-278  
    Code Editor, 62  
    downloading, 6  
    “Hello, World” example, 53-58  
    projects, creating, 54, 60  
    UI (user interface), 61, 67-91  
        adding solutions and projects, 70-73  
        Code Editor, 86-89  
        customizing, 83-85  
        parts of, 68  
        solution and project properties, 74-82  
        windows, creating, 55-56  
    visual transitions, 791-792  
    visual tree, 506, 519, 767

## W

Watch window, 101  
waterfall models, 20  
WCF (Windows Communication Foundation), 488  
While keyword, 192-194  
widgets, adding to windows, 55-56. *See also* WPF

- controls
- width of panel controls, 543
- Window class, 308
- Window control, 558-562
- windows. *See also* UI (user interface) in Visual Studio
  - arranging, 84
  - creating, 55-56
  - logical tree, 766
- Windows Forms (WinForms), 491
- WithEvents keyword, 477
- WPF (Windows Presentation Foundation), 501
  - animations, 747-755
    - class hierarchy, 748
    - example of, 752-755
    - reasons for using, 747
    - storyboards, 749-750
    - timeline classes, 751
  - bindings, 797
    - binding modes, 805
    - binding source, 807-810
    - to collections, 814-842
    - creating, 802-804
    - DataContext property, 811
    - reasons for using, 799-800
    - structure of, 798
    - update triggers, 806
  - class hierarchy, 535
  - control templates, 721, 765, 775-787
    - control contracts, 784-787
    - example of, 776-778
    - necessary control components, 783
    - presenters, 781-782
    - syntax, 775
    - template bindings, 779-780
    - visual states, 788-794
    - visual transitions, 791-792
  - dependency properties, 591
    - callbacks, 614-623
    - capabilities of, 595
    - classes, 602-603
    - creating, 604-609
    - registering, 610-612
    - reusing, 613
    - value calculation, 596-601
  - graphics, 669-671
    - brushes, 677-697
    - color, 674-676
    - effects, 712-714
    - flow documents, 708-711
    - pens, 698-702
    - typography, 703-707
  - hierarchies in, 506, 519-522
  - resources, 719, 721
    - advantages and disadvantages of, 722
    - in code, 729
    - defining, 724-726
    - referencing, 727-728
  - routed commands, 629-630, 654-666
    - class hierarchy, 663
    - creating, 665
    - in FCL (Framework Command Library), 656-658
    - hooking up, 660-662
    - input gestures, 664-665
    - logical design, 655, 659
  - routed events, 627, 632-653
    - arguments, 635-644
    - creating, 648-653
    - dependency properties versus, 628, 645
    - reasons for using, 632-633
    - strategies, 634
  - styles, 597, 721, 730-735
    - hierarchies in, 734-735
    - types of, 730-731
  - themes, 597
  - transformations, 756-761
  - triggers, 721, 736-746
    - event triggers, 744-746
    - property triggers, 736-743
    - types of, 736
- WPF controls, 531-533
  - application creation example, 578-587
  - building, 769-774
  - content controls, 557-569
    - buttons, 565-568
    - headered content controls, 563-564
    - Window control, 558-562
  - control classes, 553-556
  - control contracts, 784-787
  - customizing. *See* graphics; templates
  - flow controls, 711
  - items controls, 570-576
    - class hierarchy, 570-571
    - ComboBox control, 575-576
    - ListBox control, 575-576
    - TreeView control, 572-574
  - necessary components, 783

- panel controls, 537-552
  - attached properties, 538-541
  - DockPanel control, 544-546
  - Grid control, 547-548
  - GridSplitter control, 549-550
  - nesting, 551
  - positioning and sizing, 542-543
- tab controls, 577
- visual tree, 767
- WPF Designer, 55-56, 61, 68
- WrapPanel control, 537
- wrapping, 593
- write-only properties, 247

## X

- XAML, 55, 501. *See also* WPF (Windows Presentation Foundation)
  - attributes and properties, 509
  - collections, 513-514
  - compilation process, 523-527
  - content properties, 510-512
  - event handlers, 639
  - example of, 505
  - namespaces, 515-516
  - need for, 504
  - object elements, 509
  - referencing objects from, 831
  - root element, 515
  - Visual Basic versus, 507-508
  - WPF trees, 519-522

## XML

- comments, 146-147
- need for, 504

## Z

- Zulu, 336