Adam Nathan

WPF 4.5

UNLEASHED

FREE SAMPLE CHAPTER

Some Praise for the First Edition of *Windows® Presentation Foundation Unleashed*

"The Nathan book is brilliant—you'll love it. Publishers, take note: I'd sure be buying a heck of a lot more technical books if more of them were like this one."
—Jeff Atwood, codinghorror.com, cofounder of Stack Overflow

"*Windows Presentation Foundation Unleashed* is a must-have book for anyone interested in learning and using WPF. Buy it, read it, and keep it close to your computer."
—Josh Smith, Microsoft MVP

"As we built the feature team that delivered the new WPF presentation layer for Visual Studio 2010, *Windows Presentation Foundation Unleashed* quickly became our must-read WPF reference book of choice, over and above other books on WPF and indeed internal documentation. Highly recommended for any developer wanting to learn how to make the most of WPF."
—James Bartlett, senior lead program manager, Microsoft Visual Studio

"I've bought nearly all available WPF books, but the only one that's still on my desk is *Windows Presentation Foundation Unleashed*. It not only covers all WPF aspects, but it does it in the right, concise way so that reading it was a real pleasure."
—Corrado Cavalli, Codeworks

"*Windows Presentation Foundation Unleashed* is the most insightful WPF book there is. Don't be misled by its size; this book has the best introduction and deepest insights. This is the must-read for anyone getting started or wanting to get the most out of WPF."
—Jaime Rodriguez, Microsoft client evangelist for Windows, WPF, Silverlight, and Windows Phone

"I found *Windows Presentation Foundation Unleashed* to be an excellent and thorough introduction and guide to programming WPF. It is clearly written, easily understood, and yet still deep enough to get a good understanding of how WPF works and how to use it. Not a simple feat to accomplish! I heartily recommend it to all the students who take DevelopMentor's WPF course! Anyone serious about doing WPF work should have a copy in their library."
—Mark Smith, DevelopMentor instructor, author of DevelopMentor's Essential WPF course

"I have read *Windows Presentation Foundation Unleashed* from cover to cover and have found it to be really the most comprehensive material on WPF. I can't think of even a single instance when I have not been able to find the solution (or a pointer to one) every time that I have picked up the book to figure out the intricacies of WPF."
—Durgesh Nayak, team leader, Axis Technical Group

"*Windows Presentation Foundation Unleashed* is the book that made WPF make so much sense for me. Without Adam's work, WPF would still be a mystery to me and my team. The enthusiasm for WPF is evident from the offset and it really rubs off on the reader."
—Peter O'Hanlon, managing director, Lifestyle Computing Ltd

"Adam Nathan's *Windows Presentation Foundation Unleashed* must surely be considered one of the seminal books on WPF. It has everything you need to help you get to grips with the learning cliff that is WPF. It certainly taught me loads, and even now, after several years of full-time WPF development, *Windows Presentation Foundation Unleashed* is never far from my hand."
—Sacha Barber, Microsoft MVP, CodeProject MVP, author of many WPF articles

"Of all the books published about WPF, there are only three that I recommend. *Windows Presentation Foundation Unleashed* is my primary recommendation to developers looking to get up to speed quickly with WPF."
—Mike Brown, Microsoft MVP, Client App Development, and president of KharaSoft, Inc.

Adam Nathan

WPF 4.5

UNLEASHED

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

WPF 4.5 Unleashed

ISBN-13: 978-0-672-33697-3
ISBN-10: 0-672-33697-9

Library of Congress Control Number: 2013939232

Printed in the United States on America

First Printing July 2013

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(s) 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 programs accompanying it.

Bulk Sales

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

U.S. Corporate and Government Sales

1-800-382-3419

corpsales@pearsontechgroup.com

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

International Sales

international@pearsoned.com

Editor-in-Chief
Greg Wiegand

Executive Editor
Neil Rowe

Development Editor
Mark Renfrow

Managing Editor
Kristy Hart

Senior Project Editor
Betsy Gratner

Indexer
Erika Millen

Proofreader
Kathy Ruiz

Technical Editors
Dwayne Need
Robert Hogue
Joe Castro
Jordan Parker

Publishing Coordinator
Cindy Teeters

Cover Designer
Mark Shirar

Senior Compositor
Gloria Schurick

Contents at a Glance

	Introduction	1
Part I	**Background**	
1	Why WPF?	7
2	XAML Demystified	17
3	WPF Fundamentals	55
Part II	**Building a WPF Application**	
4	Sizing, Positioning, and Transforming Elements	77
5	Layout with Panels	97
6	Input Events: Keyboard, Mouse, Stylus, and Touch	141
7	Structuring and Deploying an Application	177
8	Exploiting Windows Desktop Features	217
Part III	**Controls**	
9	Content Controls	241
10	Items Controls	255
11	Images, Text, and Other Controls	309
Part IV	**Features for Professional Developers**	
12	Resources	341
13	Data Binding	361
14	Styles, Templates, Skins, and Themes	415
Part V	**Rich Media**	
15	2D Graphics	473
16	3D Graphics	535
17	Animation	605
18	Audio, Video, and Speech	651
Part VI	**Advanced Topics**	
19	Interoperability with Non-WPF Technologies	671
20	User Controls and Custom Controls	717
21	Layout with Custom Panels	747
22	Toast Notifications	771
A	Fun with XAML Readers and Writers	783
	Index	799

Table of Contents

Introduction **1**

Who Should Read This Book? 2
Software Requirements 3
Code Examples 3
How This Book Is Organized 4
Conventions Used in This Book 6

Part I Background

1 Why WPF? **7**

A Look at the Past 8
Enter WPF 9
The Evolution of WPF 12
Summary 16

2 XAML Demystified **17**

XAML Defined 19
Elements and Attributes 20
Namespaces 22
Property Elements 25
Type Converters 26
Markup Extensions 28
Children of Object Elements 31
Mixing XAML with Procedural Code 36
XAML2009 44
XAML Keywords 49
Summary 52

3 WPF Fundamentals **55**

A Tour of the Class Hierarchy 55
Logical and Visual Trees 57
Dependency Properties 62
Summary 76

Part II Building a WPF Application

4 Sizing, Positioning, and Transforming Elements 77

Controlling Size ... 78
Controlling Position ... 83
Applying Transforms ... 86
Summary ... 95

5 Layout with Panels 97

Canvas ... 98
StackPanel ... 100
WrapPanel ... 102
DockPanel ... 105
Grid ... 108
Primitive Panels ... 120
Handling Content Overflow ... 122
Putting It All Together: Creating a Visual Studio–Like Collapsible, Dockable, Resizable Pane ... 130
Summary ... 140

6 Input Events: Keyboard, Mouse, Stylus, and Touch 141

Routed Events ... 141
Keyboard Events ... 150
Mouse Events ... 152
Stylus Events ... 156
Touch Events ... 158
Commands ... 170
Summary ... 176

7 Structuring and Deploying an Application 177

Standard Desktop Applications ... 177
Navigation-Based Desktop Applications ... 193
Gadget-Style Applications ... 205
XAML Browser Applications ... 207
Loose XAML Pages ... 213
Summary ... 215

8 Exploiting Windows Desktop Features 217

Jump Lists ... 217
Taskbar Item Customizations ... 229
Aero Glass ... 233
TaskDialog ... 236
Summary ... 239

Part III Controls

9 Content Controls 241

Buttons 243
Simple Containers 248
Containers with Headers 252
Summary 254

10 Items Controls 255

Common Functionality 256
Selectors 261
Menus 298
Other Items Controls 302
Summary 308

11 Images, Text, and Other Controls 309

The Image Control 309
Text and Ink Controls 311
Documents 318
Range Controls 334
Calendar Controls 336
Summary 340

Part IV Features for Professional Developers

12 Resources 341

Binary Resources 341
Logical Resources 349
Summary 360

13 Data Binding 361

Introducing the Binding Object 361
Controlling Rendering 373
Customizing the View of a Collection 385
Data Providers 396
Advanced Topics 403
Putting It All Together: The Pure-XAML Twitter Client 412
Summary 414

14 Styles, Templates, Skins, and Themes 415

Styles 416
Templates 430
Skins 458
Themes 465
Summary 470

Part V Rich Media

15 2D Graphics 473

Drawings 474
Visuals 491
Shapes 503
Brushes 511
Effects 527
Improving Rendering Performance 530
Summary 533

16 3D Graphics 535

Getting Started with 3D Graphics 536
Cameras and Coordinate Systems 540
Transform3D 552
Model3D 561
Visual3D 584
Viewport3D 591
2D and 3D Coordinate System Transformation 594
Summary 603

17 Animation 605

Animations in Procedural Code 606
Animations in XAML 619
Keyframe Animations 628
Easing Functions 635
Animations and the Visual State Manager 641
Summary 649

18 Audio, Video, and Speech 651

Audio 651
Video 656
Speech 662
Summary 669

Part VI Advanced Topics

19 Interoperability with Non-WPF Technologies 671

Embedding Win32 Controls in WPF Applications ... 673
Embedding WPF Controls in Win32 Applications ... 688
Embedding Windows Forms Controls in WPF Applications ... 695
Embedding WPF Controls in Windows Forms Applications ... 700
Mixing DirectX Content with WPF Content ... 704
Embedding ActiveX Controls in WPF Applications ... 710
Summary ... 714

20 User Controls and Custom Controls 717

Creating a User Control ... 719
Creating a Custom Control ... 728
Summary ... 746

21 Layout with Custom Panels 747

Communication Between Parents and Children ... 748
Creating a SimpleCanvas ... 751
Creating a SimpleStackPanel ... 756
Creating an OverlapPanel ... 759
Creating a FanCanvas ... 764
Summary ... 769

22 Toast Notifications 771

Prerequisites ... 771
Sending a Toast Notification ... 774
Toast Templates ... 775
Notification Events ... 778
Scheduled Notifications ... 779
Summary ... 780

A Fun with XAML Readers and Writers 783

Overview ... 783
The Node Loop ... 786
Reading XAML ... 787
Writing to Live Objects ... 791
Writing to XML ... 793
XamlServices ... 794

Index 799

About the Author

Adam Nathan is a principal software architect for Microsoft in the Startup Business Group. Adam was previously the founding developer and architect for Popfly, Microsoft's first product built on Silverlight, named one of the 25 most innovative products of 2007 by *PCWorld Magazine*. Having started his career on Microsoft's Common Language Runtime team, Adam has been at the core of .NET and WPF technologies since the very beginning.

Adam's books have been considered required reading by many inside Microsoft and throughout the industry. He is the author of the best-selling *WPF Unleashed* (Sams, 2006) that was nominated for a 2008 Jolt Award, *WPF 4 Unleashed* (Sams, 2010), *Windows 8 Apps with XAML and C# Unleashed* (Sams, 2012), *101 Windows Phone 7 Apps* (Sams, 2011), *Silverlight 1.0 Unleashed* (Sams, 2008), and *.NET and COM: The Complete Interoperability Guide* (Sams, 2002); a coauthor of *ASP.NET: Tips, Tutorials, and Code* (Sams, 2001); and a contributor to books including *.NET Framework Standard Library Annotated Reference, Volume 2* (Addison-Wesley, 2005) and *Windows Developer Power Tools* (O'Reilly, 2006). Adam is also the creator of PINVOKE.NET and its Visual Studio add-in. You can find him online at www.adamnathan.net or @adamnathan on Twitter.

Dedication

To Tyler and Ryan.

Acknowledgments

Although most of the process of writing a book is very solitary, this book came together because of the work of many talented and hard-working people. I'd like to take a moment to thank some of them by name.

I'd like to sincerely thank Dwayne Need, senior development manager from the WPF team. His feedback on my drafts was so thorough and insightful, the book is far better because of him. I'd like to thank Robert Hogue, Joe Castro, and Jordan Parker for their helpful reviews. David Teitlebaum, 3D expert from the WPF team, deserves many thanks for agreeing to update the great 3D chapter originally written by Daniel Lehenbauer. Having Daniel's and David's perspectives and advice captured on paper is a huge benefit for any readers thinking about dabbling in 3D.

I'd also like to thank (in alphabetical order): Chris Brumme, Eileen Chan, Brian Chapman, Beatriz de Oliveira Costa, Joe Duffy, Ifeanyi Echeruo, Dan Glick, Neil Kronlage, Rico Mariani, Mike Mueller, Oleg Ovetchkine, Lori Pearce, S. Ramini, Rob Relyea, Tim Rice, Ben Ronco, Eric Rudder, Adam Smith, Tim Sneath, David Treadwell, and Paramesh Vaidyanathan.

I'd like to thank the folks at Sams—especially Neil Rowe and Betsy Gratner, who are always a pleasure to work with. I couldn't have asked for a better publishing team. Never once was I told that my content was too long or too short or too different from a typical *Unleashed* title. They gave me the complete freedom to write the kind of book I wanted to write.

I'd like to thank my mom, dad, and brother for opening my eyes to the world of computer programming when I was in elementary school. If you have children, please expose them to the magic of writing software while they're still young enough to care about what you have to say!

Finally, I thank *you* for picking up a copy of this book and reading at least this far! I hope you continue reading and find the journey of exploring WPF 4.5 as fascinating as I have!

We Want to Hear from You!

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

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

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

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

E-mail: feedback@samspublishing.com

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

Reader Services

Visit our website and register this book at 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

Thank you for picking up *WPF 4.5 Unleashed*! Windows Presentation Foundation (WPF) is Microsoft's premier technology for creating Windows desktop apps, whether they consist of plain forms, document-centric windows, animated cartoons, videos, immersive 3D environments, or all of the above. WPF is a technology that makes it easier than ever to create a broad range of applications. It's also the basis for XAML-based Windows Store apps.

Ever since WPF was publicly announced ten years ago (with the code name "Avalon"), it has gotten considerable attention for the ways in which it revolutionizes the process of creating software—especially for Windows programmers used to Windows Forms and GDI. It's relatively easy to create fun, useful, and shareable WPF samples that demonstrate all kinds of techniques that are difficult to accomplish in other technologies. WPF 4.5, released in August 2012, continues to improve on previous versions of WPF in many different dimensions.

WPF is quite a departure from previous technologies in terms of its programming model, underlying concepts, and basic terminology. Even viewing the source code for WPF (by cracking open its components with a tool such as .NET Reflector) is a confusing experience because the code you're looking for often doesn't reside where you'd expect to find it. When you combine all this with the fact that there are often several ways to accomplish any task in WPF, you arrive at a conclusion shared by many: *WPF has a very steep learning curve*.

That's where this book comes in. As WPF was developed, it was obvious that there would be no shortage of WPF books in the marketplace. But it wasn't clear to me that the books would have the right balance to guide people through the technology and its unique concepts while showing practical ways to exploit it. Therefore, I wrote the first edition of this book, *Windows Presentation Foundation Unleashed*, with the following goals in mind:

- To provide a solid grounding in the underlying concepts, in a practical and approachable fashion
- To answer the questions most people have when learning the technology and to show how commonly desired tasks are accomplished
- To be an authoritative source, thanks to input from members of the WPF team who designed, implemented, and tested the technology
- To be clear about where the technology falls short rather than selling the technology as the answer to all problems
- To be an easily navigated reference that you can constantly come back to

The first two editions of this book were far more successful than I ever imagined they would be. Now, more than six years after the first edition, I believe that this book accomplishes all the same goals but with even more depth and in the context of the modern

Windows experience that includes Windows Store apps and a different design aesthetic. Whether you're new to WPF or a long-time WPF developer, I hope you find this book to exhibit all these attributes.

Who Should Read This Book?

This book is for software developers who are interested in creating user interfaces for the Windows desktop. Regardless of whether you're creating line-of-business applications, consumer-facing applications, or reusable controls, this book contains a lot of content that helps you get the most out of the platform. It's designed to be understandable even for folks who are new to the .NET Framework. And if you are already well versed in WPF, I'm confident that this book still has information for you. At the very least, it should be an invaluable reference for your bookshelf.

Because the technology and concepts behind WPF are the same ones behind Silverlight and XAML-based Windows Store apps, reading this book can also make you a better developer for Windows Phone and the Windows Store.

Although this book's content is not optimized for graphic designers, reading this book can be a great way to understand more of the "guts" behind a product like Blend for Visual Studio.

To summarize, this book does the following:

- Covers everything you need to know about Extensible Application Markup Language (XAML), the XML-based language for creating declarative user interfaces that can be easily restyled
- Examines the WPF feature areas in incredible depth: controls, layout, resources, data binding, styling, graphics, animation, and more
- Delves into topics that aren't covered by most books: 3D, speech, audio/video, documents, effects, and more
- Shows how to create popular user interface elements and leverage built-in controls such as the new Office-style Ribbon
- Demonstrates how to create sophisticated user interface mechanisms, such as Visual Studio–like collapsible/dockable panes
- Explains how to develop and deploy all types of applications, including navigation-based applications, applications hosted in a web browser, and applications with great-looking nonrectangular windows
- Explains how to create first-class custom controls for WPF
- Demonstrates how to create hybrid WPF software that leverages Windows Forms, DirectX, ActiveX, or other non-WPF technologies
- Explains how to exploit desktop features in WPF applications, such as Jump Lists and taskbar customizations, and the same toast notifications used by Windows Store apps

This book doesn't cover every last bit of WPF. (In particular, XML Paper Specification [XPS] documents, which never really took off, are given only a small bit of attention.) WPF's surface area is so large that I don't believe any single book can. But I think you'll be pleased with the breadth and depth achieved by this book.

Examples in this book appear in XAML and C#, plus C++/CLI for interoperability discussions. XAML is used heavily for a number of reasons: It's often the most concise way to express source code, it can often be pasted into lightweight tools to see instant results without any compilation, WPF-based tools generate XAML rather than procedural code, and XAML is applicable no matter what .NET language you use, such as Visual Basic instead of C#. Whenever the mapping between XAML and a language such as C# is not obvious, examples are shown in both representations.

Software Requirements

This book targets version 4.5 of Windows Presentation Foundation, the corresponding Windows SDK, and Visual Studio 2012.

The following software is required:

- A version of Windows that supports the .NET Framework 4.5.
- The .NET Framework 4.5, which is installed by default starting with Windows 8. For earlier versions of Windows, you can download the .NET Framework 4.5 for free from http://msdn.com.

In addition, the following software is recommended:

- The Windows Software Development Kit (SDK), specifically the .NET tools it includes. This is also a free download from http://msdn.com.
- Visual Studio 2012 or later, which can be a free Express edition downloaded from http://msdn.com.

If you want additional tool support for WPF-based graphic design, Blend for Visual Studio can be extremely helpful.

A few examples are specific to features introduced in Windows Vista, Windows 7, and Windows 8. Some examples require a touchscreen (or an equivalent touch digitizer). The rest of the book applies equally to all relevant versions of Windows.

Code Examples

The source code for examples in this book can be downloaded from http://informit.com/title/9780672336973 or http://adamnathan.net/wpf.

How This Book Is Organized

This book is arranged into six main parts, representing the progression of feature areas that you typically need to understand to use WPF effectively. But if you're dying to jump ahead and learn about a topic such as 3D or animation, the book is set up to allow for nonlinear journeys as well. The following sections provide a summary of each part.

Part I: Background

This part includes the following chapters:

- Chapter 1: Why WPF?
- Chapter 2: XAML Demystified
- Chapter 3: WPF Fundamentals

Chapter 1 introduces WPF by comparing it to alternative technologies and helping you make decisions about when WPF is appropriate for your needs. Chapter 2 explores XAML in great depth, giving you the foundation to understand the XAML you'll encounter in the rest of the book and in real life. Chapter 3 highlights the most unique pieces of WPF's programming model above and beyond what .NET programmers already understand.

Part II: Building a WPF Application

This part includes the following chapters:

- Chapter 4: Sizing, Positioning, and Transforming Elements
- Chapter 5: Layout with Panels
- Chapter 6: Input Events: Keyboard, Mouse, Stylus, and Touch
- Chapter 7: Structuring and Deploying an Application
- Chapter 8: Exploiting Windows Desktop Features

Part II equips you with the knowledge to assemble and deploy a traditional-looking application (although some fancier effects, such as transforms and nonrectangular windows, are also covered). Chapters 4 and 5 discuss arranging controls (and other elements) in a user interface. Chapter 6 covers input events, including support for engaging touch user interfaces. Chapter 7 examines several different ways to package and deploy WPF-based user interfaces to make complete applications. Chapter 8 ends this part by showing slick ways to exploit features in the Windows desktop that can help make your application look modern.

Part III: Controls

This part includes the following chapters:

- Chapter 9: Content Controls
- Chapter 10: Items Controls
- Chapter 11: Images, Text, and Other Controls

Part III provides a tour of controls built into WPF. There are many that you'd expect to have available, plus several that you might not expect. Two categories of controls—content controls (Chapter 9) and items controls (Chapter 10)—are important and deep enough topics to merit their own chapters. The rest of the controls are examined in Chapter 11.

Part IV: Features for Professional Developers

This part includes the following chapters:

- Chapter 12: Resources
- Chapter 13: Data Binding
- Chapter 14: Styles, Templates, Skins, and Themes

The features covered in Part IV are not always necessary to use in WPF applications, but they can greatly enhance the development process. Therefore, they are indispensable for professional developers who are serious about creating maintainable and robust applications or components. These topics are less about the results visible to end users than they are about the best practices for accomplishing these results.

Part V: Rich Media

This part includes the following chapters:

- Chapter 15: 2D Graphics
- Chapter 16: 3D Graphics
- Chapter 17: Animation
- Chapter 18: Audio, Video, and Speech

This part of the book covers the features in WPF that typically get the most attention. The support for 2D and 3D graphics, animation, video, and more enable you to create a stunning experience. These features—and the way they are exposed—set WPF apart from previous systems. WPF lowers the barrier to incorporating such content in your software, so you might try some of these features that you never would have dared to try in the past!

Part VI: Advanced Topics

This part includes the following chapters:

- Chapter 19: Interoperability with Non-WPF Technologies
- Chapter 20: User Controls and Custom Controls
- Chapter 21: Layout with Custom Panels
- Chapter 22: Toast Notifications

The topics covered in Part VI are relevant for advanced application developers, or developers of WPF-based controls. The fact that existing WPF controls can be radically restyled greatly reduces the need for creating custom controls. The final chapter is especially interesting because it enables your WPF apps to use a feature designed for Windows Store apps.

Conventions Used in This Book

Various typefaces in this book identify new terms and other special items. These typefaces include the following:

Typeface	Meaning
Italic	Italic is used for new terms or phrases when they are initially defined and occasionally for emphasis.
`Monospace`	`Monospace` is used for screen messages, code listings, and command samples, as well as filenames. In code listings, *`italic monospace type`* is used for placeholder text. Code listings are colorized similar to the way they are colorized in Visual Studio. `Blue monospace type` is used for XML elements and C#/C++ keywords, `brown monospace type` is used for XML element names and C#/C++ strings, `green monospace type` is used for comments, `red monospace type` is used for XML attributes, and `teal monospace type` is used for type names in C# and C++.

Throughout this book, you'll find a number of sidebar elements:

FAQ

What is a FAQ sidebar?

A FAQ sidebar presents a question readers might have regarding the subject matter in a particular spot in the book—and then provides a concise answer.

DIGGING DEEPER

A Digging Deeper sidebar presents advanced or more detailed information on a subject than is provided in the surrounding text. Think of Digging Deeper material as stuff you can look into if you're curious but can ignore if you're not.

TIP

A tip is a bit of information that can help you in a real-world situation. Tips often offer shortcuts or alternative approaches to produce better results or to make a task easier or quicker.

WARNING

A warning alerts you to an action or a condition that can lead to an unexpected or unpredictable result—and then tells you how to avoid it.

This page intentionally left blank

CHAPTER 3

WPF Fundamentals

IN THIS CHAPTER

- A Tour of the Class Hierarchy
- Logical and Visual Trees
- Dependency Properties

To finish Part I, "Background," and before moving on to the *really* fun topics, it's helpful to examine some of the main concepts that WPF introduces above and beyond what .NET programmers are already familiar with. The topics in this chapter are some of the main culprits responsible for WPF's notoriously steep learning curve. By familiarizing yourself with these concepts now, you'll be able to approach the rest of this book (or any other WPF documentation) with confidence.

Some of this chapter's concepts are unique to WPF (such as logical and visual trees), but others are just extensions of concepts that should be quite familiar (such as properties). As you learn about each one, you'll see how to apply it to a very simple piece of user interface that most programs need—an *About dialog*.

A Tour of the Class Hierarchy

WPF's classes have a very deep inheritance hierarchy, so it can be hard to get your head wrapped around the significance of various classes and their relationships. A handful of classes are fundamental to the inner workings of WPF and deserve a quick explanation before we get any further in the book. Figure 3.1 shows these important classes and their relationships.

These 12 classes have the following significance:

- **`Object`**—The base class for all .NET classes and the only class in the figure that isn't WPF specific.

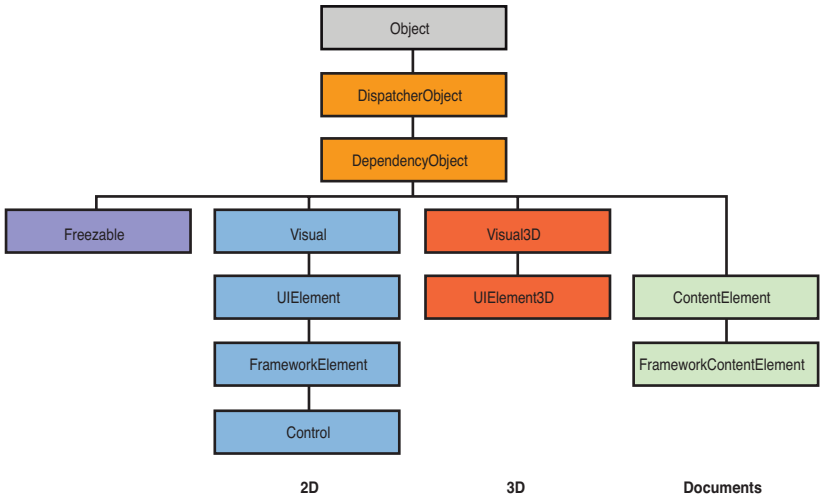

FIGURE 3.1 The core classes that form the foundation of WPF.

- **`DispatcherObject`**—The base class meant for any object that wishes to be accessed only on the thread that created it. Most WPF classes derive from `DispatcherObject` and are therefore inherently thread-unsafe. The `Dispatcher` part of the name refers to WPF's version of a Win32-like message loop, discussed further in Chapter 7, "Structuring and Deploying an Application."
- **`DependencyObject`**—The base class for any object that can support dependency properties, one of the main topics in this chapter.
- **`Freezable`**—The base class for objects that can be "frozen" into a read-only state for performance reasons. `Freezables`, once frozen, can be safely shared among multiple threads, unlike all other `DispatcherObjects`. Frozen objects can never be unfrozen, but you can clone them to create unfrozen copies. Most `Freezables` are graphics primitives such as brushes, pens, and geometries or animation classes.
- **`Visual`**—The base class for all objects that have their own 2D visual representation. `Visuals` are discussed in depth in Chapter 15, "2D Graphics."
- **`UIElement`**—The base class for all 2D visual objects with support for routed events, command binding, layout, and focus. These features are discussed in Chapter 5, "Layout with Panels," and Chapter 6, "Input Events: Keyboard, Mouse, Stylus, and Touch."
- **`Visual3D`**—The base class for all objects that have their own 3D visual representation. `Visual3Ds` are discussed in depth in Chapter 16, "3D Graphics."
- **`UIElement3D`**—The base class for all 3D visual objects with support for routed events, command binding, and focus, also discussed in Chapter 16.
- **`ContentElement`**—A base class similar to `UIElement` but for document-related pieces of content that don't have rendering behavior on their own. Instead,

ContentElements are hosted in a Visual-derived class to be rendered on the screen. Each ContentElement often requires multiple Visuals to render correctly (spanning lines, columns, and pages).

- **FrameworkElement**—The base class that adds support for styles, data binding, resources, and a few common mechanisms for controls, such as tooltips and context menus.
- **FrameworkContentElement**—The analog to FrameworkElement for content. Chapter 11, "Images, Text, and Other Controls," examines the FrameworkContentElements in WPF.
- **Control**—The base class for familiar controls such as Button, ListBox, and StatusBar. Control adds many properties to its FrameworkElement base class, such as Foreground, Background, and FontSize, as well as the ability to be completely restyled. Part III, "Controls," examines WPF's controls in depth.

Throughout the book, the simple term *element* is used to refer to an object that derives from UIElement or FrameworkElement, and sometimes ContentElement or FrameworkContentElement. The distinction between UIElement and FrameworkElement or between ContentElement and FrameworkContentElement is not important because WPF doesn't ship any other public subclasses of UIElement and ContentElement.

Logical and Visual Trees

XAML is natural for representing a user interface because of its hierarchical nature. In WPF, user interfaces are constructed from a tree of objects known as a *logical tree*.

Listing 3.1 defines the beginnings of a hypothetical About dialog, using a Window as the root of the logical tree. The Window has a StackPanel child element (described in Chapter 5) containing a few simple controls plus another StackPanel that contains Buttons.

LISTING 3.1 A Simple About Dialog in XAML

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  Title="About WPF 4.5 Unleashed" SizeToContent="WidthAndHeight"
  Background="OrangeRed">
  <StackPanel>
    <Label FontWeight="Bold" FontSize="20" Foreground="White">
      WPF 4.5 Unleashed
    </Label>
    <Label>© 2013 SAMS Publishing</Label>
    <Label>Installed Chapters:</Label>
    <ListBox>
      <ListBoxItem>Chapter 1</ListBoxItem>
      <ListBoxItem>Chapter 2</ListBoxItem>
    </ListBox>
```

LISTING 3.1 Continued

```
    <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
      <Button MinWidth="75" Margin="10">Help</Button>
      <Button MinWidth="75" Margin="10">OK</Button>
    </StackPanel>
    <StatusBar>You have successfully registered this product.</StatusBar>
  </StackPanel>
</Window>
```

Figure 3.2 shows the rendered dialog (which you can easily produce by pasting the content of Listing 3.1 into a tool such as the XAMLPAD2009 sample from the previous chapter), and Figure 3.3 illustrates the logical tree for this dialog.

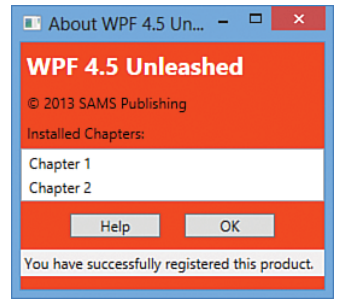

FIGURE 3.2 The rendered dialog from Listing 3.1.

Note that a logical tree exists even for WPF user interfaces that aren't created in XAML. Listing 3.1 could be implemented entirely in C#, and the logical tree would be identical.

The logical tree concept is straightforward, but why should you care about it? Because just about every aspect of WPF (properties, events, resources, and so on) has behavior tied to the logical tree. For example, property values are sometimes propagated down the tree to child elements automatically, and raised events can travel up or down the tree. This behavior of property values is discussed later in this chapter, and this behavior of events is discussed in Chapter 6.

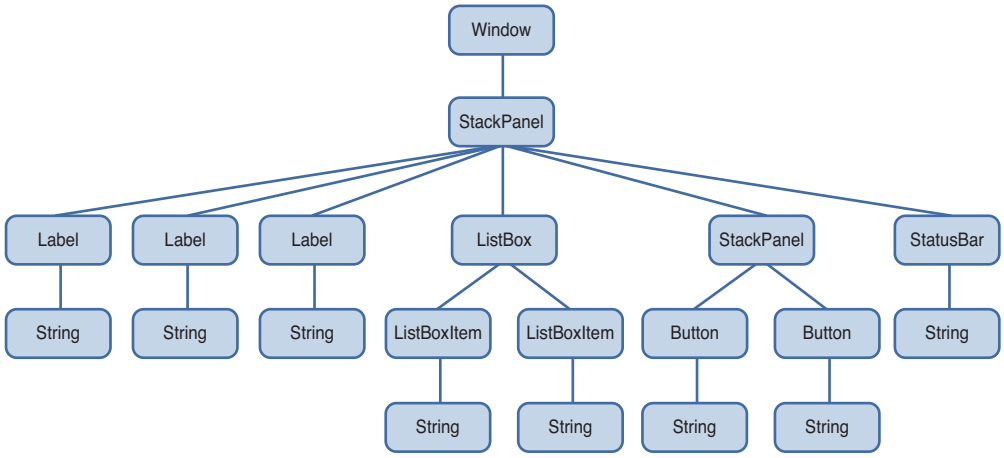

FIGURE 3.3 The logical tree for Listing 3.1.

The logical tree exposed by WPF is a simplification of what is actually going on when the elements are rendered. The entire tree of elements actually being rendered is called the

visual tree. You can think of the visual tree as an expansion of a logical tree, in which nodes are broken down into their core visual components. Rather than leaving each element as a "black box," a visual tree exposes the visual implementation details. For example, although a `ListBox` is logically a single control, its default visual representation is composed of more primitive WPF elements: a `Border`, two `ScrollBars`, and more.

Not all logical tree nodes appear in the visual tree; only the elements that derive from `System.Windows.Media.Visual` or `System.Windows.Media.Visual3D` are included. Other elements (and simple string content, as in Listing 3.1) are not included because they don't have inherent rendering behavior of their own.

TIP

Some lightweight XAML viewers, such as the XamlPadX tool mentioned in the preceding chapter, have functionality for exploring the visual tree (and property values) for the objects that it renders from XAML.

Figure 3.4 illustrates the default visual tree for Listing 3.1. This diagram exposes some inner components of the user interface that are currently invisible, such as the `ListBox`'s two `ScrollBars` and each `Label`'s `Border`. It also reveals that `Button`, `Label`, and `ListBoxItem` are all composed of the same elements. (These controls have other visual differences as the result of different default property values. For example, `Button` has a default `Margin` of `10` on all sides, whereas `Label` has a default `Margin` of `0`.)

Because they enable you to peer inside the deep composition of WPF elements, visual trees can be surprisingly complex. Fortunately, although visual trees are an essential part of the WPF infrastructure, you often don't need to worry about them unless you're radically restyling controls (covered in Chapter 14, "Styles, Templates, Skins, and Themes") or doing low-level drawing (covered in Chapter 15). Writing code that depends on a specific visual tree for a `Button`, for example, breaks one of WPF's core tenets—the separation of look and logic. When someone restyles a control such as `Button` using the techniques described in Chapter 14, its entire visual tree is replaced with something that could be completely different.

WARNING

Avoid writing code that depends on a specific visual tree!

Whereas a logical tree is static without programmer intervention (such as dynamically adding/removing elements), a visual tree can change simply because a user switches to a different Windows theme!

However, you can easily traverse both the logical and visual trees using the somewhat symmetrical `System.Windows.LogicalTreeHelper` and `System.Windows.Media.VisualTreeHelper` classes. Listing 3.2 contains a code-behind file for Listing 3.1 that, when run under a debugger, outputs a simple depth-first representation of both the logical and visual trees for the About dialog. (This requires adding `x:Class="AboutDialog"` and the corresponding `xmlns:x` directive to Listing 3.1 in order to hook it up to this procedural code.)

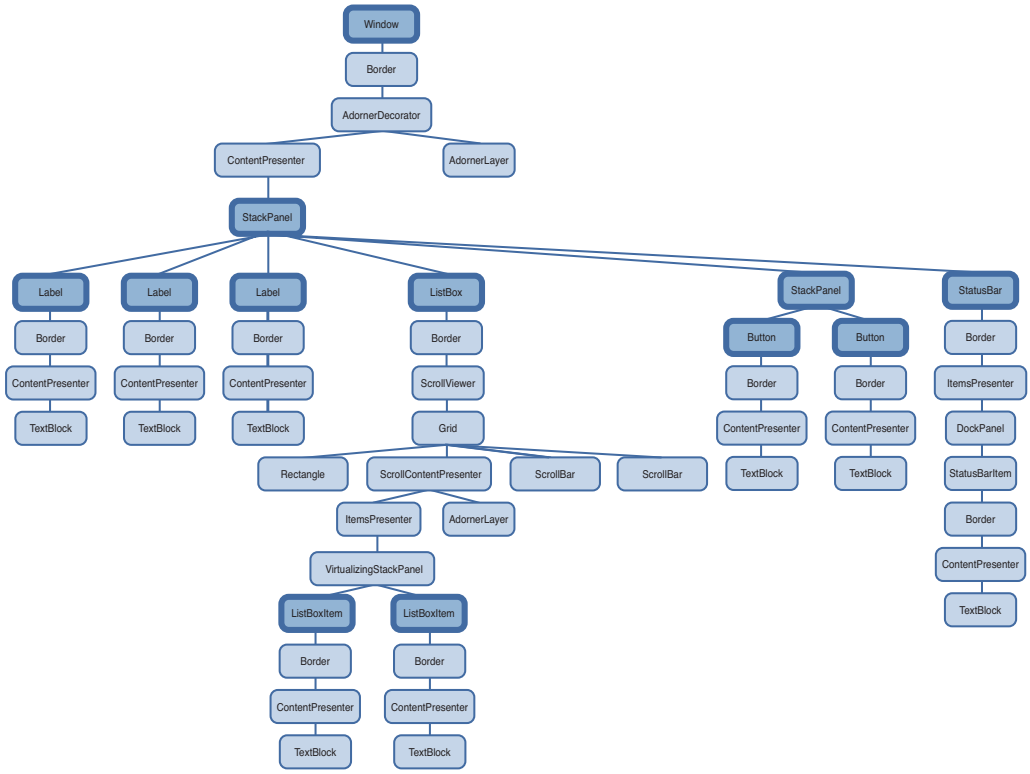

FIGURE 3.4 The visual tree for Listing 3.1, with logical tree nodes emphasized.

LISTING 3.2 Walking and Printing the Logical and Visual Trees

```
using System;
using System.Diagnostics;
using System.Windows;
using System.Windows.Media;

public partial class AboutDialog : Window
{
  public AboutDialog()
  {
    InitializeComponent();
    PrintLogicalTree(0, this);
  }

  protected override void OnContentRendered(EventArgs e)
  {
    base.OnContentRendered(e);
    PrintVisualTree(0, this);
```

LISTING 3.2 Continued

```
  }

  void PrintLogicalTree(int depth, object obj)
  {
    // Print the object with preceding spaces that represent its depth
    Debug.WriteLine(new string(' ', depth) + obj);

    // Sometimes leaf nodes aren't DependencyObjects (e.g. strings)
    if (!(obj is DependencyObject)) return;

    // Recursive call for each logical child
    foreach (object child in LogicalTreeHelper.GetChildren(
      obj as DependencyObject))
      PrintLogicalTree(depth + 1, child);
  }

  void PrintVisualTree(int depth, DependencyObject obj)
  {
    // Print the object with preceding spaces that represent its depth
    Debug.WriteLine(new string(' ', depth) + obj);

    // Recursive call for each visual child
    for (int i = 0; i < VisualTreeHelper.GetChildrenCount(obj); i++)
      PrintVisualTree(depth + 1, VisualTreeHelper.GetChild(obj, i));
  }
}
```

When calling these methods with a depth of 0 and the current `Window` instance, the result is a text-based tree with exactly the same nodes shown in Figures 3.2 and 3.3. Although the logical tree can be traversed within `Window`'s constructor, the visual tree is empty until the `Window` undergoes layout at least once. That is why `PrintVisualTree` is called within `OnContentRendered`, which doesn't get called until after layout occurs.

TIP

Visual trees like the one represented in Figure 3.4 are often referred to simply as *element trees*, because they encompass both elements in the logical tree and elements specific to the visual tree. The term *visual tree* is then used to describe any subtree that contains visual-only (illogical?) elements. For example, most people would say that `Window`'s default visual tree consists of a `Border`, an `AdornerDecorator`, an `AdornerLayer`, a `ContentPresenter`, and nothing more. In Figure 3.4, the top-most `StackPanel` is generally *not* considered to be the visual child of the `ContentPresenter`, despite the fact that `VisualTreeHelper` presents it as one.

TIP

In the Visual Studio debugger, you can click the little magnifying glass next to an instance of a `Visual`-derived variable to invoke WPF Visualizer, as shown in Figure 3.5. This enables you to navigate and visualize the visual tree.

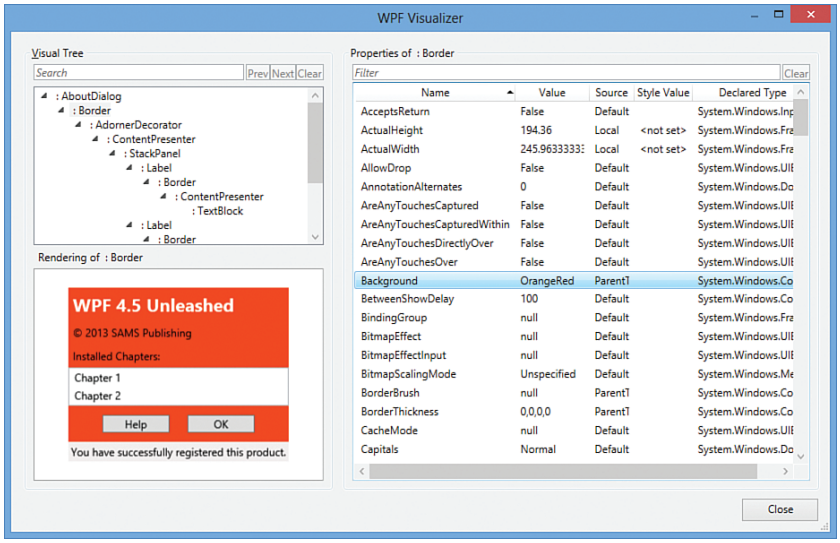

FIGURE 3.5 The WPF Visualizer in Visual Studio reveals the visual tree and details about each element.

Navigating either tree can sometimes be done with instance methods on the elements themselves. For example, the `Visual` class contains three protected members (`VisualParent`, `VisualChildrenCount`, and `GetVisualChild`) for examining its visual parent and children. `FrameworkElement`, the common base class for controls such as `Button` and `Label`, and its peer `FrameworkContentElement` both define a public `Parent` property representing the logical parent and a protected `LogicalChildren` property for the logical children. Subclasses of these two classes often publicly expose their logical children in a variety of ways, such as in a public `Children` collection. Some classes, such as `Button` and `Label`, expose a `Content` property and enforce that the element can have only one logical child.

Dependency Properties

WPF introduces a type of property called a *dependency property* that is used throughout the platform to enable styling, automatic data binding, animation, and more. You might first meet this concept with skepticism, as it complicates the picture of .NET types having simple fields, properties, methods, and events. But when you understand the problems that dependency properties solve, you will likely accept them as a welcome addition.

A dependency property *depends* on multiple providers for determining its value at any point in time. These providers could be an animation continuously changing its value, a parent element whose property value propagates down to its children, and so on. Arguably the biggest feature of a dependency property is its built-in ability to provide change notification.

The motivation for adding such intelligence to properties is to enable rich functionality directly from declarative markup. The key to WPF's declarative-friendly design is its heavy use of properties. `Button`, for example, has more than 100 public properties (most of which are inherited from `Control` and its base classes)! Properties can be easily set in XAML (directly or by using a design tool) without any procedural code. But without the extra plumbing in dependency properties, it would be hard for the simple action of setting properties to get the desired results without the need to write additional code.

In this section, we briefly look at the implementation of a dependency property to make this discussion more concrete, and then we dig deeper into some of the ways that dependency properties add value on top of plain .NET properties:

- Change notification
- Property value inheritance
- Support for multiple providers

Understanding most of the nuances of dependency properties is usually important only for custom control authors. However, even casual users of WPF need to be aware of what dependency properties are and how they work. For example, you can only style and animate dependency properties. After working with WPF for a while, you might find yourself wishing that *all* properties would be dependency properties!

A Dependency Property Implementation

In practice, dependency properties are just normal .NET properties hooked into some extra WPF infrastructure. This is all accomplished via WPF APIs; no .NET languages (other than XAML) have an intrinsic understanding of a dependency property.

Listing 3.3 demonstrates how `Button` effectively implements one of its dependency properties, called `IsDefault`.

LISTING 3.3 A Standard Dependency Property Implementation

```
public class Button : ButtonBase
{
  // The dependency property
  public static readonly DependencyProperty IsDefaultProperty;

  static Button()
  {
    // Register the property
```

```
    Button.IsDefaultProperty = DependencyProperty.Register("IsDefault",
      typeof(bool), typeof(Button),
      new FrameworkPropertyMetadata(false,
      new PropertyChangedCallback(OnIsDefaultChanged)));
    …
  }

  // A .NET property wrapper (optional)
  public bool IsDefault
  {
    get { return (bool)GetValue(Button.IsDefaultProperty); }
    set { SetValue(Button.IsDefaultProperty, value); }
  }

  // A property changed callback (optional)
  private static void OnIsDefaultChanged(
    DependencyObject o, DependencyPropertyChangedEventArgs e) { … }
  …
}
```

The static `IsDefaultProperty` field is the actual dependency property, represented by the `System.Windows.DependencyProperty` class. By convention, all `DependencyProperty` fields are public, static, and have a `Property` suffix. Several pieces of infrastructure require that you follow this convention: localization tools, XAML loading, and more.

Dependency properties are usually created by calling the static `DependencyProperty.Register` method, which requires a name (`IsDefault`), a property type (`bool`), and the type of the class claiming to own the property (`Button`). Optionally (via different overloads of `Register`), you can pass metadata that customizes how the property is treated by WPF, as well as callbacks for handling property value changes, coercing values, and validating values. `Button` calls an overload of `Register` in its static constructor to give the dependency property a default value of `false` and to attach a delegate for change notifications.

Finally, the traditional .NET property called `IsDefault` implements its accessors by calling `GetValue` and `SetValue` methods inherited from `System.Windows.DependencyObject`, the low-level base class from which all classes with dependency properties must derive. `GetValue` returns the last value passed to `SetValue` or, if `SetValue` has never been called, the default value registered with the property.

The `IsDefault` .NET property (sometimes called a *property wrapper* in this context) is not strictly necessary; consumers of `Button` could directly call the `GetValue`/`SetValue` methods because they are exposed publicly. But the .NET property makes programmatic reading

TIP

Visual Studio has a snippet called `propdp` that automatically expands into a definition of a dependency property, which makes defining one much faster than doing all the typing yourself!

and writing of the property much more natural for consumers, and it enables the property to be set via XAML. WPF should, but does not, provide generic overloads of `GetValue` and `SetValue`. This is primarily because dependency properties were invented before .NET generics were widely used.

WARNING

.NET property wrappers are bypassed at runtime when setting dependency properties in XAML!

Although the XAML compiler depends on the property wrapper at compile time, WPF calls the underlying `GetValue` and `SetValue` methods directly at runtime! Therefore, to maintain parity between setting a property in XAML and procedural code, it's crucial that property wrappers not contain any logic in addition to the `GetValue`/`SetValue` calls. If you want to add custom logic, that's what the registered callbacks are for. All of WPF's built-in property wrappers abide by this rule, so this warning is for anyone writing a custom class with its own dependency properties.

On the surface, Listing 3.3 looks like an overly verbose way of representing a simple Boolean property. However, because `GetValue` and `SetValue` internally use an efficient sparse storage system and because `IsDefaultProperty` is a static field (rather than an instance field), the dependency property implementation saves per-instance memory compared to a typical .NET property. If all the properties on WPF controls were wrappers around instance fields (as most .NET properties are), they would consume a significant amount of memory because of all the local data attached to each instance. Having more than 100 fields for each `Button`, more than 100 fields for each `Label`, and so forth would add up quickly! Instead, almost all of `Button`'s and `Label`'s properties are dependency properties.

The benefits of the dependency property implementation extend to more than just memory usage, however. The implementation centralizes and standardizes a fair amount of code that property implementers would have to write to check thread access, prompt the containing element to be re-rendered, and so on. For example, if a property requires its element to be re-rendered when its value changes (such as `Button`'s `Background` property), it can simply pass the `FrameworkPropertyMetadataOptions.AffectsRender` flag to an overload of `DependencyProperty.Register`. In addition, this implementation enables the three features listed earlier that we'll now examine one-by-one, starting with change notification.

Change Notification

Whenever the value of a dependency property changes, WPF can automatically trigger a number of actions, depending on the property's metadata. These actions can be re-rendering the appropriate elements, updating the current layout, refreshing data bindings, and much more. One of the interesting features enabled by this built-in change notification is *property triggers*, which enable you to perform your own custom actions when a property value changes, without writing any procedural code.

For example, imagine that you want the text in each `Button` from the About dialog in Listing 3.1 to turn blue when the mouse pointer hovers over it. Without property triggers, you can attach two event handlers to each `Button`, one for its `MouseEnter` event and one for its `MouseLeave` event:

```
<Button MouseEnter="Button_MouseEnter" MouseLeave="Button_MouseLeave"
        MinWidth="75" Margin="10">Help</Button>
<Button MouseEnter="Button_MouseEnter" MouseLeave="Button_MouseLeave"
        MinWidth="75" Margin="10">OK</Button>
```

These two handlers could be implemented in a C# code-behind file as follows:

```
// Change the foreground to blue when the mouse enters the button
void Button_MouseEnter(object sender, MouseEventArgs e)
{
  Button b = sender as Button;
  if (b != null) b.Foreground = Brushes.Blue;
}

// Restore the foreground to black when the mouse exits the button
void Button_MouseLeave(object sender, MouseEventArgs e)
{
  Button b = sender as Button;
  if (b != null) b.Foreground = Brushes.Black;
}
```

With a property trigger, however, you can accomplish this same behavior purely in XAML. The following concise `Trigger` object is just about all you need:

```
<Trigger Property="IsMouseOver" Value="True">
  <Setter Property="Foreground" Value="Blue"/>
</Trigger>
```

This trigger can act on `Button`'s `IsMouseOver` property, which becomes `true` at the same time the `MouseEnter` event is raised and `false` at the same time the `MouseLeave` event is raised. Note that you don't have to worry about reverting `Foreground` to black when `IsMouseOver` changes to `false`. This is automatically done by WPF!

The only trick is assigning this `Trigger` to each `Button`. Unfortunately, because of a confusing limitation, you can't apply property triggers directly to elements such as `Button`. You can apply them only inside a `Style` object, so an in-depth examination of property triggers is saved for Chapter 14. In the meantime, to experiment with property triggers, you can apply the preceding `Trigger` to a `Button` by wrapping it in a few intermediate XML elements, as follows:

```
<Button MinWidth="75" Margin="10">
<Button.Style>
  <Style TargetType="{x:Type Button}">
```

```
    <Style.Triggers>
      <Trigger Property="IsMouseOver" Value="True">
        <Setter Property="Foreground" Value="Blue"/>
      </Trigger>
    </Style.Triggers>
    </Style>
</Button.Style>
    OK
</Button>
```


Property triggers are just one of three types of triggers supported by WPF. A *data trigger* is a form of property trigger that works for all .NET properties (not just dependency properties), also covered in Chapter 14. An *event trigger* enables you to declaratively specify actions to take when a routed event (covered in Chapter 6) is raised. Event triggers always involve working with animations or sounds, so they aren't covered until Chapter 17, "Animation."

WARNING

Don't be fooled by an element's `Triggers` collection!

`FrameworkElement`'s `Triggers` property is a read/write collection of `TriggerBase` items (the common base class for all three types of triggers), so it looks like an easy way to attach property triggers to controls such as `Button`. Unfortunately, this collection can only contain event triggers, so its name and type are misleading. Attempting to add a property trigger (or data trigger) to the collection causes an exception to be thrown at runtime.

Property Value Inheritance

The term *property value inheritance* (or *property inheritance* for short) doesn't refer to traditional object-oriented class-based inheritance but rather the flowing of property values down the element tree. A simple example of this can be seen in Listing 3.4, which updates the `Window` from Listing 3.1 by explicitly setting its `FontSize` and `FontStyle` dependency properties. Figure 3.6 shows the result of this change. (Notice that the `Window` automatically resizes to fit all the content thanks to its slick `SizeToContent` setting!)

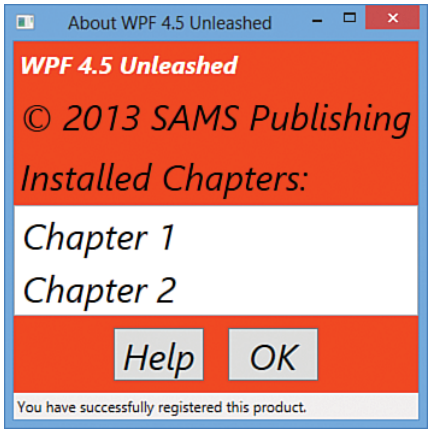

FIGURE 3.6 The About dialog with `FontSize` and `FontStyle` set on the root `Window`.

LISTING 3.4 The About Dialog with Font Properties Set on the Root Window

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  Title="About WPF 4.5 Unleashed" SizeToContent="WidthAndHeight"
  FontSize="30" FontStyle="Italic"
  Background="OrangeRed">
  <StackPanel>
    <Label FontWeight="Bold" FontSize="20" Foreground="White">
      WPF 4.5 Unleashed
    </Label>
    <Label>© 2013 SAMS Publishing</Label>
    <Label>Installed Chapters:</Label>
    <ListBox>
      <ListBoxItem>Chapter 1</ListBoxItem>
      <ListBoxItem>Chapter 2</ListBoxItem>
    </ListBox>
    <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
      <Button MinWidth="75" Margin="10">Help</Button>
      <Button MinWidth="75" Margin="10">OK</Button>
    </StackPanel>
    <StatusBar>You have successfully registered this product.</StatusBar>
  </StackPanel>
</Window>
```

For the most part, these two settings flow all the way down the tree and are inherited by children. This affects even the `Buttons` and `ListBoxItems`, which are three levels down the logical tree. The first `Label`'s `FontSize` does not change because it is explicitly marked with a `FontSize` of `20`, overriding the inherited value of `30`. The inherited `FontStyle` setting of `Italic` affects all `Labels`, `ListBoxItems`, and `Buttons`, however, because none of them have this set explicitly.

Notice that the text in the `StatusBar` is unaffected by either of these values, despite the fact that it supports these two properties just like the other controls. The behavior of property value inheritance can be subtle in cases like this for two reasons:

- Not every dependency property participates in property value inheritance. (Internally, dependency properties can opt in to inheritance by passing `FrameworkPropertyMetadataOptions.Inherits` to `DependencyProperty.Register`.)
- There may be other higher-priority sources setting the property value, as explained in the next section.

In this case, the latter reason is to blame. A few controls, such as `StatusBar`, `Menu`, and `ToolTip`, internally set their font properties to match current system settings. This way, users get the familiar experience of controlling their font via Control Panel. The result can be confusing, however, because such controls end up "swallowing" any inheritance from

proceeding further down the element tree. For example, if you add a `Button` as a logical child of the `StatusBar` in Listing 3.4, its `FontSize` and `FontStyle` would be the default values of `12` and `Normal`, respectively, unlike the other `Buttons` outside of the `StatusBar`.

DIGGING DEEPER

Property Value Inheritance in Additional Places

Property value inheritance was originally designed to operate on the element tree, but it has been extended to work in a few other contexts as well. For example, values can be passed down to certain elements that *look like* children in the XML sense (because of XAML's property element syntax) but *are not* children in terms of the logical or visual trees. These pseudochildren can be an element's triggers or the value of *any* property (not just `Content` or `Children`), as long as it is an object deriving from `Freezable`. This may sound arbitrary and isn't well documented, but the intention is that several XAML-based scenarios "just work" as you would expect, without requiring you to think about it.

Support for Multiple Providers

WPF contains many powerful mechanisms that independently attempt to set the value of dependency properties. Without a well-defined mechanism for handling these disparate property value providers, the system would be a bit chaotic, and property values could be unstable. Of course, as their name indicates, dependency properties were designed to depend on these providers in a consistent and orderly manner.

Figure 3.7 illustrates the five-step process that WPF runs each dependency property through in order to calculate its final value. This process happens automatically, thanks to the built-in change notification in dependency properties.

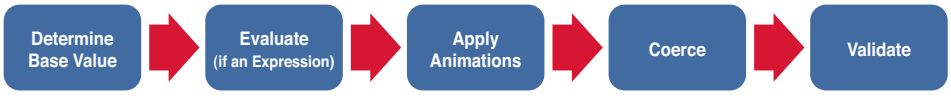

FIGURE 3.7 The pipeline for calculating the value of a dependency property.

Step 1: Determine the Base Value

Most of the property value providers factor into the base value calculation. The following list reveals the ten providers that can set the value of most dependency properties, in order from highest to lowest precedence:

1. Local value
2. Parent template trigger
3. Parent template
4. Style triggers
5. Template triggers

6. Style setters
7. Theme style triggers
8. Theme style setters
9. Property value inheritance
10. Default value

You've already seen some of the property value providers, such as property value inheritance (#9). *Local value* (#1) technically means any call to `DependencyObject.SetValue`, but this is typically seen with a simple property assignment in XAML or procedural code (because of the way dependency properties are implemented, as shown previously with `Button.IsDefault`). *Default value* (#10) refers to the initial value registered with the dependency property, which naturally has the lowest precedence. The other providers, which all involve styles and templates, are explained further in Chapter 14.

This order of precedence explains why `StatusBar`'s `FontSize` and `FontStyle` were not impacted by property value inheritance in Listing 3.4. The setting of `StatusBar`'s font properties to match system settings is done via theme style setters (#8). Although this has precedence over property value inheritance (#9), you can still override these font settings using any mechanism with a higher precedence, such as simply setting local values on `StatusBar`.

TIP

If you can't figure out where a given dependency property is getting its current value, you can use the static `DependencyPropertyHelper.GetValueSource` method as a debugging aid. This returns a `ValueSource` structure that contains a few pieces of data: a `BaseValueSource` enumeration that reveals where the base value came from (step 1 in the process) and Boolean `IsExpression`, `IsAnimated`, and `IsCoerced` properties that reveal information about steps 2 through 4.

When calling this method on the `StatusBar` instance from Listing 3.1 or 3.4 with the `FontSize` or `FontStyle` property, the returned `BaseValueSource` is `DefaultStyle`, revealing that the value comes from a theme style setter. (Theme styles are sometimes referred to as *default styles*. The enumeration value for a theme style trigger is `DefaultStyleTrigger`.)

Do *not* use this method in production code! Future versions of WPF could break assumptions you've made about the value calculation. In addition, treating a property value differently, depending on its source, goes against the way things are supposed to work in WPF applications.

DIGGING DEEPER

Clearing a Local Value

The earlier "Change Notification" section demonstrates the use of procedural code to change a `Button`'s `Foreground` to blue in response to the `MouseEnter` event and then changing it back to black in response to the `MouseLeave` event. The problem with this approach is that black is set as a local value inside `MouseLeave`, which is much different from the `Button`'s initial state, in which its black `Foreground` comes from a setter in its theme style. If the theme is changed and the new theme tries to change the default `Foreground` color (or if other providers with higher precedence try to do the same), this change is trumped by the local setting of black.

What you likely want to do instead is *clear* the local value and let WPF set the value from the relevant provider with the next-highest precedence. Fortunately, `DependencyObject` provides exactly this kind of mechanism with its `ClearValue` method. This can be called on a `Button` `b` as follows in C#:

```
b.ClearValue(Button.ForegroundProperty);
```

(`Button.ForegroundProperty` is the static `DependencyProperty` field.) After calling `ClearValue`, the local value is simply removed from the equation when WPF recalculates the base value.

Note that the trigger on the `IsMouseOver` property from the "Change Notification" section does not have the same problem as the implementation with event handlers. A trigger is either active or inactive, and when it is inactive, it is simply ignored in the property value calculation.

Step 2: Evaluate

If the value from step one is an *expression* (an object deriving from `System.Windows.Expression`), WPF performs a special evaluation step to convert the expression into a concrete result. Expressions mostly appear in data binding (the topic of Chapter 13, "Data Binding").

Step 3: Apply Animations

If one or more animations are running, they have the power to alter the current property value (using the value after step 2 as input) or completely replace it. Therefore, animations (the topic of Chapter 17) can trump all other property value providers—even local values! This is often a stumbling block for people who are new to WPF.

Step 4: Coerce

After all the property value providers have had their say, WPF passes the almost-final property value to a `CoerceValueCallback` delegate, if one was registered with the dependency property. The callback is responsible for returning a new value, based on custom logic. For example, built-in WPF controls such as `ProgressBar` use this callback to constrain its `Value` dependency property to a value between its `Minimum` and `Maximum` values, returning `Minimum` if the input value is less than `Minimum` and `Maximum` if the input value is greater than `Maximum`. If you change your coercion logic at runtime, you can call `CoerceValue` to make WPF run the new coercion and validation logic again.

Step 5: Validate

Finally, the potentially coerced value is passed to a `ValidateValueCallback` delegate, if one was registered with the dependency property. This callback must return `true` if the input value is valid and `false` otherwise. Returning `false` causes an exception to be thrown, canceling the entire process.

TIP

`DependencyObject` has a `SetCurrentValue` method that directly updates the current value of a property without changing its value source. (The value is still subject to coercion and validation.) This is meant for controls that set values in response to user interaction. For example, the `RadioButton` control modifies the value of the `IsChecked` property on other `RadioButtons` in the same group, based on user interaction.

Attached Properties

An *attached property* is a special form of dependency property that can effectively be *attached* to arbitrary objects. This may sound strange at first, but this mechanism has several applications in WPF.

For the About dialog example, imagine that rather than setting `FontSize` and `FontStyle` for the entire `Window` (as is done in Listing 3.4), you would rather set them on the inner `StackPanel` so they are inherited only by the two `Buttons`. Moving the property attributes to the inner `StackPanel` element doesn't work, however, because `StackPanel` doesn't have any font-related properties of its own! Instead, you must use the `FontSize` and `FontStyle` attached properties that happen to be defined on a class called `TextElement`. Listing 3.5 demonstrates this, introducing new XAML syntax designed especially for attached properties. This enables the desired property value inheritance, as shown in Figure 3.8.

LISTING 3.5 The About Dialog with Font Properties Moved to the Inner `StackPanel`

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  Title="About WPF 4.5 Unleashed" SizeToContent="WidthAndHeight"
  Background="OrangeRed">
  <StackPanel>
    <Label FontWeight="Bold" FontSize="20" Foreground="White">
      WPF 4.5 Unleashed
    </Label>
    <Label>© 2013 SAMS Publishing</Label>
    <Label>Installed Chapters:</Label>
    <ListBox>
      <ListBoxItem>Chapter 1</ListBoxItem>
      <ListBoxItem>Chapter 2</ListBoxItem>
    </ListBox>
```

```
    <StackPanel TextElement.FontSize="30" TextElement.FontStyle="Italic"
      Orientation="Horizontal" HorizontalAlignment="Center">
      <Button MinWidth="75" Margin="10">Help</Button>
      <Button MinWidth="75" Margin="10">OK</Button>
    </StackPanel>
    <StatusBar>You have successfully registered this product.</StatusBar>
  </StackPanel>
</Window>
```

`TextElement.FontSize` and `TextElement.FontStyle` (rather than simply `FontSize` and `FontStyle`) must be used in the `StackPanel` element because `StackPanel` does not have these properties. When a XAML parser or compiler encounters this syntax, it requires that `TextElement` (sometimes called the *attached property provider*) have static methods called `SetFontSize` and `SetFontStyle` that can set the value accordingly. Therefore, the `StackPanel` declaration in Listing 3.5 is equivalent to the following C# code:

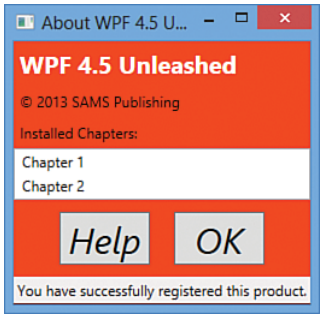

FIGURE 3.8 The About dialog with `FontSize` and `FontStyle` set on both `Buttons` via inheritance from the inner `StackPanel`.

```
StackPanel panel = new StackPanel();
TextElement.SetFontSize(panel, 30);
TextElement.SetFontStyle(panel, FontStyles.Italic);
panel.Orientation = Orientation.Horizontal;
panel.HorizontalAlignment =
HorizontalAlignment.Center;
Button helpButton = new Button();
helpButton.MinWidth = 75;
helpButton.Margin = new Thickness(10);
helpButton.Content = "Help";
Button okButton = new Button();
okButton.MinWidth = 75;
okButton.Margin = new Thickness(10);
okButton.Content = "OK";
panel.Children.Add(helpButton);
panel.Children.Add(okButton);
```

Notice that the enumeration values such as `FontStyles.Italic`, `Orientation.Horizontal`, and `HorizontalAlignment.Center` were previously specified in XAML simply as `Italic`, `Horizontal`, and `Center`, respectively. This is possible thanks to the `EnumConverter` type converter in the .NET Framework, which can convert any case-insensitive string.

Although the XAML in Listing 3.5 nicely represents the logical attachment of `FontSize` and `FontStyle` to `StackPanel`, the C# code reveals that there's no real magic here, just a method call that associates an element with an otherwise-unrelated property. One of the interesting things about the attached property abstraction is that no .NET property is a part of it!

Internally, methods such as `SetFontSize` simply call the same `DependencyObject.SetValue` method that a normal dependency property accessor calls, but on the passed-in `DependencyObject` rather than the current instance:

```
public static void SetFontSize(DependencyObject element, double value)
{
  element.SetValue(TextElement.FontSizeProperty, value);
}
```

Similarly, attached properties also define a static `Get`*`XXX`* method (where *`XXX`* is the name of the property) that calls the familiar `DependencyObject.GetValue` method:

```
public static double GetFontSize(DependencyObject element)
{
  return (double)element.GetValue(TextElement.FontSizeProperty);
}
```

As with property wrappers for normal dependency properties, these `Get`*`XXX`* and `Set`*`XXX`* methods must not do anything other than make a call to `GetValue` and `SetValue`.

DIGGING DEEPER

Understanding the Attached Property Provider

The most confusing part about the `FontSize` and `FontStyle` attached properties used in Listing 3.5 is that they aren't defined by `Button` or even `Control`, the base class that defines the normal `FontSize` and `FontStyle` dependency properties! Instead, they are defined by the seemingly unrelated `TextElement` class (and also by the `TextBlock` class, which could alternatively be used in the preceding examples).

How can this possibly work when `TextElement.FontSizeProperty` is a separate `DependencyProperty` field from `Control.FontSizeProperty` (and `TextElement.FontStyleProperty` is separate from `Control.FontStyleProperty`)? The key is the way these dependency properties are internally registered. If you were to look at the source code for `TextElement`, you would see something like the following:

```
TextElement.FontSizeProperty = DependencyProperty.RegisterAttached(
  "FontSize", typeof(double), typeof(TextElement), new FrameworkPropertyMetadata(
  SystemFonts.MessageFontSize, FrameworkPropertyMetadataOptions.Inherits |
  FrameworkPropertyMetadataOptions.AffectsRender |
  FrameworkPropertyMetadataOptions.AffectsMeasure),
  new ValidateValueCallback(TextElement.IsValidFontSize));
```

This is similar to the earlier example of registering `Button`'s `IsDefault` dependency property, except that the `RegisterAttached` method optimizes the handling of property metadata for attached property scenarios.

`Control`, on the other hand, doesn't register its `FontSize` dependency property! Instead, it calls `AddOwner` on `TextElement`'s already-registered property, getting a reference to exactly the same instance:

Continued

```
Control.FontSizeProperty = TextElement.FontSizeProperty.AddOwner(
  typeof(Control), new FrameworkPropertyMetadata(SystemFonts.MessageFontSize,
  FrameworkPropertyMetadataOptions.Inherits));
```

Therefore, the `FontSize`, `FontStyle`, and other font-related dependency properties inherited by all controls *are* the same properties exposed by `TextElement`!

Fortunately, in most cases, the class that exposes an attached property (the `Get`*XXX* and `Set`*XXX* methods) is the same class that defines the normal dependency property, avoiding this confusion.

DIGGING DEEPER

Attached Properties as an Extensibility Mechanism

As in previous technologies such as Windows Forms, many classes in WPF define a `Tag` property (of type `System.Object`) intended for storing arbitrary custom data with each instance. But attached properties are a more powerful and flexible mechanism for attaching custom data to any object deriving from `DependencyObject`. It's often overlooked that attached properties even enable you to effectively add custom data to instances of sealed classes (and WPF has plenty of them)!

A further twist to the story of attached properties is that although setting them in XAML relies on the presence of the static `Set`*XXX* method, you can bypass this method in procedural code and call `DependencyObject.SetValue` directly. This means that you can use *any* dependency property as an attached property in procedural code. For example, the following code attaches `ItemsControl`'s `IsTextSearchEnabled` property to a `Button` and assigns it a value:

```
// Attach an unrelated property to a Button and set its value to true:
okButton.SetValue(ItemsControl.IsTextSearchEnabledProperty, true);
```

Although this seems nonsensical, and it certainly doesn't magically enable new functionality on this `Button`, you have the freedom to consume this property value in a way that makes sense to your application or component.

There are more interesting ways to extend elements in this manner. For example, `FrameworkElement`'s `Tag` property is a dependency property, so you can attach it to an instance of `GeometryModel3D` (a class you'll see again in Chapter 16, that is sealed and does *not* have a `Tag` property), as follows:

```
GeometryModel3D model = new GeometryModel3D();
model.SetValue(FrameworkElement.TagProperty, "my custom data");
```

This is just one of the ways in which WPF provides extensibility without the need for traditional inheritance.

Although the About dialog example uses attached properties for advanced property value inheritance, attached properties are most commonly used for layout of user interface elements. (In fact, attached properties were originally designed for WPF's layout system.) Various `Panel`-derived classes define attached properties designed to be attached to their children for controlling how they are arranged. This way, each `Panel` can apply its own custom behavior to arbitrary children without requiring all possible child elements to be burdened with their own set of relevant properties. It also enables systems such as layout to be easily extensible, because anyone can write a new `Panel` with custom attached properties. Chapter 5, "Layout with Panels," and Chapter 21, "Layout with Custom Panels," have all the details.

Summary

In this chapter and the preceding two chapters, you've learned about all the major ways that WPF builds on top of the foundation of the .NET Framework. The WPF team could have exposed its features via typical .NET APIs, as in Windows Forms, and still have created an interesting technology. Instead, the team added several fundamental concepts that enable a wide range of features to be exposed in a way that can provide great productivity for developers and designers.

Indeed, when you focus on these core concepts, as this chapter does, you can see that the landscape isn't quite as simple as it used to be: There are multiple types of properties, multiple trees, and multiple ways of achieving the same results (such as writing declarative versus procedural code)! Hopefully you can now appreciate some of the value of these new mechanisms. Throughout the rest of the book, these concepts generally fade into the background as we focus on accomplishing specific development tasks.

This page intentionally left blank

Index

Symbols

\ (backslash), 30

{ } (curly braces), 29-30, 375

2D graphics

- 2D and 3D coordinate system transformation
  - explained, 594
  - Visual.TransformToAncestor method, 594-598
  - Visual3D.TransformToAncestor method, 598-603
  - Visual3D.TransformToDescendant method, 598-603
- Brushes
  - BitmapCacheBrush class, 533
  - DrawingBrush class, 518-522
  - explained, 511
  - ImageBrush class, 522-523
  - LinearGradientBrush class, 513-516
  - as opacity masks, 525-527
  - RadialGradientBrush class, 517-518
  - SolidColorBrush class, 512
  - VisualBrush class, 523-525
- drawings
  - clip art example, 489-490
  - Drawing class, 474
  - DrawingBrush class, 475
  - DrawingContext methods, 492
  - DrawingImage class, 475-477
  - DrawingVisual class, 475
  - GeometryDrawing class, 474-475
  - GlyphRunDrawing class, 474

ImageDrawing class, 474-476
Pen class, 487-489
VideoDrawing class, 474
effects, 527-529
explained, 473-474
geometries
aggregate geometries, 481
Bézier curves, 478
CombinedGeometry class, 484-485
defined, 477
EllipseGeometry class, 477
GeometryGroup class, 482-484
LineGeometry class, 477
PathGeometry class, 477-481
RectangleGeometry class, 477
representing as strings, 485-487
StreamGeometry class, 481
house example, 536
mapping to 3D, 539, 588-589
Shapes
clip art based on Shapes, 510-511
Ellipse class, 506
explained, 503-504
how they work, 507
Line class, 507-508
overuse of, 505
Path class, 509-510
Polygon class, 509
Polyline class, 508
Rectangle class, 505-506
transforms. *See* transforms
Visuals
custom rendering, 497
displaying on screen, 494-496
DrawingContext methods, 492
DrawingVisuals, 491-494
explained, 491
visual hit testing, 497-503
WPF 3.5 enhancements, 13

3D graphics
2D and 3D coordinate system transformation
explained, 594
Visual.TransformToAncestor method, 594-598
Visual3D.TransformToAncestor method, 598-603
Visual3D.TransformToDescendant method, 598-603
3D hit testing, 590-591
Cameras
blind spots, 543
coordinate systems, 540-542
explained, 540
LookDirection property, 542-546
MatrixCamera, 551
OrthographicCamera versus PerspectiveCamera, 549-551
Position property, 541-542
Transform property, 547
UpDirection property, 546-548
Z-fighting, 543
coordinate systems, 540-542
explained, 535-536
hardware acceleration
explained, 10
GDI and, 11
house example, 536-538
Lights, 540
Materials
AmbientMaterial, 573
combining, 576
DiffuseMaterial, 570-573
EmissiveMaterial, 574
explained, 569
SpecularMaterial, 575-576
Model3Ds
explained, 561
GeometryModel3D, 569
Lights, 561-568
Model3DGroup class, 582-584

pixel boundaries, 15
resolution independence, 10
texture coordinates, 582
Transform3Ds
combining, 560
explained, 552-553
house drawing example, 553-554
MatrixTransform3D class, 552, 560
RotateTransform3D class, 552, 557-560
ScaleTransform3D class, 552, 555-557
Transform3DGroup class, 552
TranslateTransform3D class, 552-555
Viewport2DVisual3D class, 588-589
Viewport3D class, 591-594
Visual3Ds
explained, 584
ModelVisual3D class, 585-586
UIElement3D class, 586-588
WPF 3.5 enhancements, 13

3D hit testing, 590-591

A

About dialog
attached events, 147-149
with font properties moved to inner StackPanel, 72-73
with font properties set on root window, 68
Help command, 173-174
initial code listing, 57-58
routed events, 144-146

absolute sizing, 113

accessing
binary resources
embedded in another assembly, 346
from procedural code, 347-348
at site of origin, 346-347
from XAML, 343-346
logical resources, 358

Action property (QueryContinueDragEventArgs class), 155

Activated event, 778-779

ActiveEditingMode property (InkCanvas control), 317

ActiveX controls, 710-714

ActualHeight property (FrameworkElements class), 80

ActualWidth property (FrameworkElements class), 80

AddBackEntry method, 200

AddHandler method, 142-143

AddToSchedule method, 779

advantages of WPF, 11

Aero Glass, 233-236

aggregate geometries, 481

AllowPartiallyTrustedCallers attribute, 210

AlternationCount property (ItemsControl class), 256

AlternationIndex property (ItemsControl class), 256

AmbientLight, 562, 567-568

AmbientMaterial class, 573

AnchoredBlock class, 326-327

AND relationships (logical), 429-430

Angle property (RotateTransform class), 88

AngleX property (SkewTransform class), 92

AngleY property (SkewTransform class), 92

animation
animation classes
AutoReverse property, 616
BeginTime property, 614-615
By property, 614
DoubleAnimation, 609-610
Duration property, 612
EasingFunction property, 618
explained, 607-608
FillBehavior property, 619
From property, 612-614
IsAdditive property, 619
IsCumulative property, 619
lack of generics, 608-609

linear interpolation, 610-611
RepeatBehavior property, 616-617
SpeedRatio property, 615
To property, 612-614
and data binding, 630
easing functions, 14
BackEase, 638
BounceEase, 638
CircleEase, 638
EasingMode property, 635
ElasticEase, 638
ExponentialEase, 638
power easing functions, 635-636
SineEase, 638
writing, 638-640
explained, 605
frame-based animation, 607
keyframe animation
discrete keyframes, 632-634
easing keyframes, 634
explained, 628
linear keyframes, 629-631
spline keyframes, 631-632
path-based animations, 635
reusing animations, 611
timer-based animation, 606-607
and Visual State Manager
button ControlTemplate with VisualStates, 641-644
transitions, 645-649
with XAMLEventTriggers/Storyboards
explained, 619-620
starting animations from property triggers, 626-627
Storyboards as Timelines, 627-628
TargetName property, 623-624
TargetProperty property, 620-623

annotations, adding to flow documents, 331-334

AnnotationService class, 331

APIs, Windows Runtime APIs, 771-772

Application class
creating applications without, 186
events, 184
explained, 181-182
Properties collection, 185
Run method, 182-183
single-instance applications, 186
Windows collection, 184

application menu (Ribbon), 291-292

ApplicationCommands class, 171

ApplicationPath property (JumpTask), 222

applications
associating Jump Lists with, 218
embedding Win32 controls in
explained, 673-674
keyboard navigation, 683-687
Webcam control, 674-683
embedding Windows Forms controls
explained, 695-696
PropertyGrid, 696-699
embedding WPF controls in
Win32 applications, 688-695
Windows Forms applications, 700-704
gadget-style applications, 205-206
loose XAML pages, 213-214
multiple-document interface (MDI), 185
navigation-based desktop applications
explained, 193-194
hyperlinks, 197-198
journal, 198-200
Navigate method, 196-197
navigation containers, 194-196
navigation events, 200-201
Page elements, 194-196
returning data from pages, 203-204
sending data to pages, 202-203
standard desktop applications
Application class, 181-186
application state, 192

ClickOnce, 192-193
common dialogs, 188-189
custom dialogs, 189-190
explained, 177-178
multithreaded applications, 187
retrieving command-line arguments in, 184
single-instance applications, 186
splash screens, 187-188
Window class, 178-180
Windows Installer, 192
XAML Browser applications
ClickOnce caching, 208
deployment, 211
explained, 207-208
full-trust XAML Browser applications, 210
integrated navigation, 210-211
limitations, 208-209
on-demand download, 212-213
security, 211
XAML Browser Applications (XBAPs), 207
Apply method, 229
AppUserModelIDs, 772-773
arbitrary objects, content and, 243
ArcSegment class, 478
Arguments keyword, 47-49
Arguments property (JumpTask), 222
ArrangeOverride method, 750-751
Array keyword, 52
associating Jump Lists with applications, 218
asynchronous data binding, 401
AsyncRecords keyword, 49
attached events, 147-149
attached properties
About dialog example, 72-74
as extensibility mechanism, 75
attached property providers, 74-75
defined, 72
attached property providers, 74-75
attenuation, 564
attributes, setting, 21
audio
embedded resources, 661
explained, 651
MediaElement, 654-656
MediaPlayer, 653-654
MediaTimeline, 654-656
SoundPlayer, 652
SoundPlayerAction class, 652-653
speech recognition
converting spoken words into text, 665-667
specifying grammar with GrammarBuilder, 668-669
specifying grammar with SRGS, 667-668
speech synthesis
explained, 662
GetInstalledVoices method, 662
PromptBuilder class, 663-665
SelectVoice method, 662
SelectVoiceByHints method, 662
SetOutputToWaveFile method, 663
SpeakAsync method, 662
Speech Synthesis Markup Language (SSML), 663-665
SpeechSynthesizer, 662
SystemSounds class, 652
toast notifications, 778
"Auto" length, 79
automation
automation IDs, 269
UI Automation, supporting in custom controls, 745-746
AutoReverse property (animation classes), 616
autosizing, 111-113
AxisAngleRotation3D class, 557-558
AxMsTscAxNotSafeForScripting control, 712-713

B

BackEase function, 638
backgrounds, setting, 359
backslash (\), 30
BAML (Binary Application Markup Language)
decompiling back into XAML, 43-44
defined, 41
Baml2006Reader class, 784
base values of dependency properties, calculating, 69-70
BaseValueSource enumeration, 70
BeginTime property (animation classes), 614-615
behavior
adding to custom controls
behavior, 733-735
code-behind file, 730
initial implementation, 729-733
resources, 730-731
creating for user controls, 721-723
Bézier curves, 478
BezierSegment class, 478
Binary Application Markup Language (BAML)
decompiling back into XAML, 43-44
defined, 41
binary resources
accessing
from procedural code, 347-348
resources at site of origin, 346-347
resources embedded in another assembly, 346
from XAML, 343-346
defining, 342-343
explained, 341
localizing
creating satellite assembly with LocBaml, 349
explained, 348
marking user interfaces with localization IDs, 349
preparing projects for multiple cultures, 348
Binding object, 361
binding
to .NET properties, 365-367
to collections, 369-372
to entire objects, 367-368
to UIElement, 368
ElementName property, 364
INotifyDataErrorInfo interface, 408-409
IsAsync property, 401
in procedural code, 361-363
RelativeSource property, 365
removing, 363
sharing source with DataContext, 372-373
StringFormat property, 374
TargetNullValue property, 364
UpdateSourceExceptionFilter property, 408
UpdateSourceTrigger property, 404
validation rules, 405-409
ValidationRules property, 406
in XAML, 363-365
binding. *See* data binding
Binding.DoNothing values, 384
BindingMode enumeration, 403
BitmapCache class, 531-533
BitmapCacheBrush class, 533
BitmapEffect, 528
bitmaps
BitmapCache class, 531-533
BitmapCacheBrush class, 533
BitmapEffect, 528
nearest-neighbor bitmap scaling, 310
WriteableBitmap class, 13
BitmapScalingMode property (RenderOptions), 306
BlackoutDates property (Calendar control), 338-339
Blend, 12
blind spots (cameras), 543
Block TextElements
AnchoredBlock class, 326-327
BlockUIContainer, 321

List, 320
Paragraph, 320
sample code listing, 321-324
Section, 320
Table, 320
BlockUIContainer Blocks, 321
BlurEffect, 527-528
Boolean keyword, 49
BooleanToVisibilityConverter, 382-383
Bottom property (Canvas), 98
BounceEase function, 638
BrushConverter type converter, 28
brushes
applying without logical resources, 350-351
BitmapCacheBrush class, 533
consolidating with logical resources, 351-353
explained, 511
ImageBrush class, 522-523
as opacity masks, 525-527
LinearGradientBrush class, 513-516
RadialGradientBrush class, 517-522
SolidColorBrush class, 512
VisualBrush class, 523-525
bubbling, 143
BuildWindowCore class, 680
built-in commands, 171-175
Button class, 63-64, 244-245
ButtonAutomationPeer class, 245
ButtonBase class, 243-244
buttons
Button class, 244-245
button ControlTemplate with VisualStates, 641-644
ButtonAutomationPeer class, 245
ButtonBase class, 243-244
CheckBox class, 246
defined, 243
RadioButton class, 246-248
RepeatButton class, 245
styling with built-in animations, 624-626
ToggleButton class, 245-246
By property (animation classes), 614
Byte keyword, 49

C

C++/CLI, 677-678
cached composition
BitmapCache class, 531-533
BitmapCacheBrush class, 533
Viewport2DVisual3D support for, 589
caching (ClickOnce), 208
Calendar control, 336-339
calendar controls
Calendar, 336-339
DatePicker, 339-340
Cameras
blind spots, 543
coordinate systems, 540-542
explained, 540
LookDirection property, 542-546
MatrixCamera, 551
OrthographicCamera versus PerspectiveCamera, 549-551
Position property, 541-542
Transform property, 547
UpDirection property, 546-548
Z-fighting, 543
CAML (Compiled Application Markup Language), 42
cancel buttons, 244
Cancel method, 167
canceling toast notifications, 780
CanExecute method, 171
CanExecuteChanged method, 171
CanUserAddRows property (DataGrid), 278
CanUserDeleteRows property (DataGrid), 278
Canvas, 98-100, 119. ***See also*** **SimpleCanvas**

capturing mouse events, 155-156
cells (DataGrid), selecting, 275
Center property (RadialGradientBrush), 517
CenterX property
RotateTransform class, 88-90
SkewTransform class, 92
CenterY property
RotateTransform class, 88-90
SkewTransform class, 92
change notification (dependency properties), 65-67
Char keyword, 50
CheckBox class, 246
child object elements
content property, 31-32
dictionaries, 33-34
lists, 32-33
processing rules, 36
values type-converted to object elements, 34
/clr compiler option, 682
CircleEase function, 638
class hierarchy, 55-57
Class keyword, 40-41, 50
ClassAttributes keyword, 50
classes. *See specific classes*
ClassModifier keyword, 50
ClearAllBindings method, 363
ClearBinding method, 363
ClearHighlightsCommand, 331
clearing
bindings, 363
local values, 71
ClearValue method, 71
CLI (Common Language Infrastructure), 677
Click event, 243-244
clickable cube example, 586-588
ClickCount property (MouseButtonEventArgs), 154
ClickMode property (ButtonBase class), 243
ClickOnce, 192-193
ClickOnce caching, 208
with unmanaged code, 193
clients, pure-XAML Twitter client, 412-414
clip art example, 489-490
clip art based on Shapes, 510-511
drawing-based implementation, 489-490
DrawingContext-based implementation, 493-494
WindowHostingVisual.cs file, 495
clipboard interaction (DataGrid), 276
ClipboardCopyMode property (DataGrid), 276
clipping, 122-124
ClipToBounds property (panels), 123
clr-namespace directive, 35
Code keyword, 50
code-behind files, 40, 730
CoerceValueCallback delegate, 71
cold start time, 187
Collapsed value (Visibility enumeration), 82
collections
binding to, 369-372
customizing collection views
creating new views, 393-395
explained, 385
filtering, 391
grouping, 387-390
navigating, 391-392
sorting, 385-387
dictionaries, 33-34
ItemsSource, 277
lists, 32-33
Properties, 185
SortDescriptions, 386
Triggers, 67
Windows, 184
CollectionViewSource class, 393
color
color brushes
applying without logical resources, 350-351
consolidating with logical resources, 351-353

LinearGradientBrush class, 513-516
RadialGradientBrush class, 517-518
SolidColorBrush class, 512
color space profiles, 513
toast notifications, 775

color brushes
applying without logical resources, 350-351
consolidating with logical resources, 351-353
LinearGradientBrush class, 513-516
RadialGradientBrush class, 517-518
SolidColorBrush class, 512

Color structure, 512

columns
DataGrid
auto-generated columns, 274-275
column types, 273-274
freezing, 277
Grid
sharing row/column sizes, 117-119
sizing, 112-116

CombinedGeometry class, 484-485

combining
Materials, 576
Transform3Ds, 560
transforms, 94

ComboBox control
ComboBoxItem objects, 266-267
customizing selection box, 262-265
events, 262
explained, 262
IsEditable property, 262
IsReadOnly property, 262
SelectionChanged event, 266

ComboBoxItem objects, 266-267

ComCtl32.dll, 238-239

command-line arguments, retrieving, 184

commands. ***See also specific commands***
built-in commands, 171-175
controls with built-in command bindings, 175-176
executing with input gestures, 175
explained, 170-171
implementing with custom controls, 741

commas in geometry strings, 487

common dialogs, 188-189

Common Language Infrastructure (CLI), 677

Compiled Application Markup Language, 42

compiling XAML, 39-41

Complete method, 167

CompleteQuadraticEase class, 640

ComponentCommands class, 172

CompositeCollection class, 410

CompositionTarget_Rendering event handler, 709

conflicting triggers, 429

ConnectionId keyword, 50

consolidating routed event handlers, 149-150

ConstantAttenuation property (PointLights), 564

containers
Expander class, 253-254
Frame class, 251-252
GroupBox class, 253
Label class, 248
navigation containers, 194-196
ToolTop class, 249-251

ContainerUIElement3D class, 588

Content build action, 342

content controls
and arbitrary objects, 243
buttons
Button class, 244-245
ButtonBase class, 243-244
CheckBox class, 246
defined, 243
RadioButton class, 246-248
RepeatButton class, 245
ToggleButton class, 245-246
containers
Expander class, 253-254
Frame class, 251-252

GroupBox class, 253
Label class, 248
ToolTip class, 249-251
ContentControl class, 242
defined, 242
content overflow, handling
clipping, 122-124
explained, 122
scaling, 126-130
scrolling, 124-126
content property, 31-32
ContentControl class, 435-437
Frame class, 252
ContentControl class, 242, 435-437
ContentElement class, 56
ContextMenu control, 301-302
ContextMenuService class, 302
contextual tabs, 294-295
ContextualTabGroupHeader property (RibbonTab), 294
Control class, 57
control parts, 740-741
control states, 741-745
control templates
editing, 457-458
explained, 430-431
mixing with styles, 456-457
named elements, 434
reusability of, 438-440
simple control template, 431-432
target type, restricting, 434-435
templated parent properties, respecting
Content property (ContentControl class), 435-437
hijacking existing properties for new purposes, 441
other properties, 438-440
triggers, 432-434
visual states
respecting with triggers, 442-446
respecting with VSM (Visual State Manager), 447-455
controls
ActiveX controls, 710-714
built-in command bindings, 175-176
buttons
Button class, 244-245
ButtonBase class, 243-244
CheckBox class, 246
defined, 243
RadioButton class, 246-248
RepeatButton class, 245
ToggleButton class, 245-246
Calendar, 336-339
ComboBox
ComboBoxItem objects, 266-267
customizing selection box, 262-265
events, 262
explained, 262
IsEditable property, 262
IsReadOnly property, 262
SelectionChanged event, 266
containers
Expander class, 253-254
Frame class, 251-252
GroupBox class, 253
Label class, 248
ToolTip class, 249-251
ContextMenu, 301-302
control parts, 447-449
control states, 449-455
custom controls, creating, 10
behavior, 729-735
code-behind file, 730
commands, 741
control parts, 740-741
control states, 741-745
explained, 717-718
generic resources, 737-738
resources, 730-731
UI Automation, 745-746

user controls versus custom controls, 718
user interfaces, 735-738
DataGrid
auto-generated columns, 274-275
CanUserAddRows property, 278
CanUserDeleteRows property, 278
clipboard interaction, 276
ClipboardCopyMode property, 276
column types, 273-274
displaying row details, 276-277
editing data, 277-278
EnableColumnVirtualization property, 276
EnableRowVirtualization property, 276
example, 272-273
freezing columns, 277
FrozenColumnCount property, 277
RowDetailsVisibilityMode property, 277
selecting rows/cells, 275
SelectionMode property, 275
SelectionUnit property, 275
virtualization, 276
DatePicker, 339-340
explained, 241-243
GridView, 270-271
InkCanvas, 316-318
ItemsControl class, 255-256
AlternationCount property, 256
AlternationIndex property, 256
DisplayMemberPath property, 256-257
HasItems property, 256
IsGrouping property, 256
IsTextSearchCaseSensitive property, 265
IsTextSearchEnabled property, 265
Items property, 255
ItemsPanel property, 256-260
ItemsSource property, 256
scrolling behavior, controlling, 260-261
ListBox
automation IDs, 269
example, 267-268
scrolling, 269
SelectionMode property, 268
sorting items in, 269
support for multiple selections, 268
ListView, 270-271
Menu, 298-301
PasswordBox, 316
ProgressBar, 335
Ribbon
application menu, 291-292
contextual tabs, 294-295
example, 278-281
galleries, 297-298
IsDropDownOpen property, 280
IsHostedInRibbonWindow property, 281
IsMinimized property, 280
key tips, 289-290
minimizing, 280
Quick Access toolbar, 292-294
resizing, 284-289
ribbon controls, 281-284, 288-289
RibbonGroup, 279, 286-288
RibbonTab, 279, 285-286
RibbonWindow, 280-281
ScreenTips, 295-296
WindowIconVisibility property, 281
RichTextBox, 316
ScrollViewer, 124-126
Selector class, 261
Slider, 335-336
StatusBar, 307-308
TabControl, 271-272
TextBlock
explained, 313
explicit versus implicit runs, 314
properties, 313
support for multiple lines of text, 315
whitespace, 314

TextBox, 315
ToolBar, 304-306
TreeView, 302-304
user controls, creating
behavior, 721-723
dependency properties, 724-727
explained, 717-718
protecting controls from accidental usage, 723-724
routed events, 727-728
user controls versus custom controls, 718
user interfaces, 719-721

ControlTemplate class
ControlTemplate with triggers, 432-434
editing, 457-458
mixing with styles, 456-457
named elements, 434
simple ControlTemplate, 431-432
TargetType property, 434-435
templated parent properties, respecting, 435-437

Convert method, 381
converting spoken words into text, 665-667
ConvertXmlStringToObjectGraph method, 795
coordinate systems, 540-542
CountToBackgroundConverter class, 380-383
CreateBitmapSourceFromHBitmap method, 704
CreateHighlightCommand, 331
CreateInkStickyNoteCommand, 331
CreateTextStickyNoteCommand, 331
CreateToastNotifier method, 775
CreateWindow method, 681
Cube example
clickable cube, 586-588
cube button style, 592-593
cube of buttons and small purple cube, 595-597
initial code listing, 583-584
TextBlocks, 599-602

culture, preparing projects for multiple cultures, 348
curly braces ({}), 29-30
CurrentItem property (ICollectionView), 391
curves, Bézier, 478
CustomCategory property (JumpTask), 223-224
customizing
advantages/disadvantages, 416
collection views
creating new views, 393-395
explained, 385
filtering, 391
grouping, 387-390
navigating, 391-392
sorting, 385-387
color space profiles, 513
controls, creating, 10
behavior, 729-735
commands, 741
control parts, 740-741
control states, 741-745
explained, 717-718
generic resources, 737-738
UI Automation, 745-746
user controls versus custom controls, 718
user interfaces, 735-738
data display, 383-384
data flow, 403-405
dialogs, 189-190
JumpTask behavior, 221-224
keyboard navigation, 306
panels
communication between parents and children, 748-751
explained, 747-748
rendering, 497
selection boxes (ComboBox control), 262-265
sorting, 387
taskbar
explained, 230
taskbar item progress bars, 230

taskbar overlays, 231
thumb buttons, 232-233
thumbnail content, 231

D

D3DImage class, 704-710
DashStyle class, 488-489
DashStyle property (Pen class), 488
data binding
animation and, 630
asynchronous data binding, 401
Binding object
binding to .NET properties, 365-367
binding to collections, 369-372
binding to entire objects, 367-368
binding to UIElement, 368
ElementName property, 364
INotifyDataErrorInfo interface, 408-409
IsAsync property, 401
in procedural code, 361-363
RelativeSource property, 365
removing, 363
sharing source with DataContext, 372-373
StringFormat property, 374
TargetNullValue property, 364
UpdateSourceExceptionFilter property, 408
UpdateSourceTrigger property, 404
validation rules, 405-409
ValidationRules property, 406
in XAML, 363-365
canceling temporarily, 384
collection views, customizing
creating new views, 393-395
explained, 385
filtering, 391
grouping, 387-390
navigating, 391-392
sorting, 385-387
CompositeCollection class, 410
data flow, customizing, 403-405
data providers
explained, 396
ObjectDataProvider class, 400-402
XmlDataProvider class, 396-400
defined, 361
Language Integrated Query (LINQ), 396
to methods, 402
MultiBinding class, 410-411
PriorityBinding class, 411-412
pure-XAML Twitter client, 412-414
rendering, controlling
data templates, 376-379
explained, 373-374
string formatting, 374-376
value converters, 380-385
troubleshooting, 383
WPF 3.5 features, 13
WPF 4.5 enhancements, 15
data flow, customizing, 403-405
Data property (DragEventArgs class), 154
data providers
explained, 396
ObjectDataProvider class, 400-402
XmlDataProvider class, 396-400
data templates, 376-379
HierarchicalDataTemplate, 398-399
template selectors, 380
data triggers, 67, 427-428
data types
bridging incompatible data types, 380-383
DateTime, 780
DateTimeOffset, 780
in XAML2009, 46
DataContext property, 372-373

DataGrid control
CanUserAddRows property, 278
CanUserDeleteRows property, 278
clipboard interaction, 276
ClipboardCopyMode property, 276
column types, 273-275
displaying row details, 276-277
editing data, 277-278
EnableColumnVirtualization property, 276
EnableRowVirtualization property, 276
example, 272-273
freezing columns, 277
FrozenColumnCount property, 277
RowDetailsVisibilityMode property, 277
selecting rows/cells, 275
SelectionMode property, 275
SelectionUnit property, 275
virtualization, 276
DataGridCheckBoxColumn, 274
DataGridComboBoxColumn, 274
DataGridHyperlinkColumn, 273
DataGridTemplateColumn, 274
DataGridTextColumn, 273
DataTrigger class, 427-428
DatePicker control, 339-340
DateTime data type, 780
DateTimeOffset data type, 780
DateValidationError event, 340
DayOfWeek enumeration, 339
DeadCharProcessedKey property (KeyEventArgs event), 150
debugger (Visual C++), 691
Decimal keyword, 50
declaration context, 373
declarative programming, 10
decorators, 127
default buttons, 244
default styles, 70
defining
binary resources, 342-343
object elements, 21
properties, 49
delegates
CoerceValueCallback, 71
delegate contravariance, 150
ValidateValueCallback, 72
DeleteStickyNotesCommand, 331
dependency properties, 419-420
adding to user controls, 724-727
attached properties
About dialog example, 72-74
as extensibility mechanism, 75
attached property providers, 74-75
defined, 72
attached property providers, 74-75
change notification, 65-67
explained, 62-63
hijacking, 441
implementation, 63-65
property triggers, 65-67
property value inheritance, 67-69
support for multiple providers
applying animations, 71
coercion, 71
determining base values, 69-70
evaluating, 71
explained, 69
validation, 72
DependencyObject class, 56, 64
DependencyPropertyHelper class, 70
deployment
ClickOnce, 192-193
Windows Installer, 192
WPF 3.5 enhancements, 14
WPF 4 enhancements, 15
XAMLBrowser applications, 211
DesiredSize property (FrameworkElements class), 79

desktop applications. *See* Windows desktop applications
desktop features. *See* Windows desktop features
DestroyWindowCore class, 680
device-independent pixels, 82
DialogFunction method, 690
dialogs
About dialog
with font properties moved to inner StackPanel, 72-73
with font properties set on root window, 68
initial code listing, 57-58
common dialogs, 188-189
custom dialogs, 189-190
dialog results, 190
modal dialogs
launching from Win32 applications, 695
launching from Windows Forms applications, 704
launching from WPF applications, 688, 699
modeless dialogs, 178
TaskDialogs, 236-239
dictionaries, 33-34, 46
DiffuseMaterial, 570-573
direct routing, 143
Direct3D, 10
Direction property
DirectionalLight, 562
PointLights, 566
DirectionalLight, 562-563
directives. *See specific directives*
DirectX
development of, 8
versus WPF, 11-12
when to use, 11-12
WPF interoperability, 13, 704-710
discrete keyframes, 632-634
Dismissed event, 779
DispatcherObject class, 56
DispatcherPriority enumeration, 187
DispatcherTimer class, 606-607
DisplayDateEnd property (Calendar control), 337
DisplayDateStart property (Calendar control), 337
displaying
flow documents, 329-331
Visuals on screen, 494-496
DisplayMemberPath property, 256-257, 369
Dock property (DockPanel), 105
DockPanel
examples, 105-107
explained, 105
interaction with child layout properties, 107-108
mimicking with Grid, 119
properties, 105
documents, flow documents
annotations, 331-334
Blocks, 320-327
creating, 318-319
defined, 318
displaying, 329-331
Inlines, 324-329
DoNothing value (Binding), 384
Double keyword, 50
DoubleAnimation class, 609-610
download groups, 212
DownloadFileGroupAsync method, 213
drag-and-drop events, 154-155
DragEventArgs class, 154-155
Drawing class, 474
DrawingBrush class, 475, 518-522
DrawingContext class
clip art example, 493-494
methods, 492
DrawingImage class, 475-477
drawings
clip art example, 489-490
Drawing class, 474
DrawingBrush class, 475
DrawingContext methods, 492

DrawingImage class, 475-477
DrawingVisual class, 475
geometries. *See* geometries
GeometryDrawing class, 474-475
GlyphRunDrawing class, 474
ImageDrawing class, 474-476
Pen class, 487-489
VideoDrawing class, 474

DrawingVisual class, 475
explained, 491
filling with content, 491-494

DropDownOpened event, 262

DropShadowEffect, 527-528

duration of animations, controlling, 612

Duration property (animation classes), 612

DwmExtendFrameIntoClientArea method, 233-235

dynamic versus static resources, 353-355

DynamicResource markup extension, 354-355

E

Ease method, 638

EaseIn method, 640-641

EaseInOut method, 640-641

easing functions, 14

easing keyframes, 634

EasingFunction property (animation classes), 618

EasingFunctionBase class, 639

EasingMode property (easing functions), 635

editing
control templates, 457-458
DataGrid data, 277-278

EditingCommands class, 172

EditingMode property (InkCanvas control), 317

EditingModeInverted property (InkCanvas control), 317

effects, 527-529

ElasticEase function, 638

element trees. *See* trees

ElementHost class, 700-702

ElementName property (Binding object), 364

elements. *See* object elements; property elements

EllipseGeometry class, 477

embedded resources, 661

EmbeddedResource build action, 343

embedding
ActiveX controls in WPF applications, 710-714
Win32 controls in WPF applications
explained, 673-674
keyboard navigation, 683-687
Webcam control, 674-683
Windows Forms controls in WPF applications
explained, 695-696
PropertyGrid, 696-699
WPF controls in Win32 applications
HwndSource class, 688-691
layout, 692-695
WPF controls in Windows Forms applications
converting between two representatives, 703-704
ElementHost class, 700-702
launching modal dialogs, 704

EmissiveMaterial class, 574

EnableClearType property (BitmapCache class), 532

EnableColumnVirtualization property (DataGrid), 276

EnableRowVirtualization property (DataGrid), 276

EnableVisualStyles method, 699

EndLineCap property (Pen class), 487

EndMember value (NodeType property), 787

EndObject value (NodeType property), 787

EndPoint property (LinearGradientBrush), 514

enumerations
BaseValueSource, 70
BindingMode, 403
DayOfWeek, 339
DispatcherPriority, 187
GeometryCombineMode, 484

GradientSpreadMethod, 515
JumpItemRejectionReason, 228
Key, 150-152
MouseButtonState, 153
PixelFormats, 311
RoutingStrategy, 143
ShutdownMode, 184
Stretch, 127
StretchDirection, 127
TileMode, 521
UpdateSourceTrigger, 404
Visibility, 82-83

EraseByPoint value (InkCanvasEditingMode), 318

EraseByStroke value (InkCanvasEditingMode), 318

error handling, 407-408

Error ProgressState, 230

ErrorsChanged event, 408

EscapePressed property (QueryContinueDragEventArgs class), 155

Euler angles, 558

EvenOdd fill (FillRule property), 480

events

attributes, 21
Click, 243-244
DateValidationError, 340
DropDownOpened, 262
event handlers, 49
event wrappers, 142
JumpItemsRejected, 228
JumpItemsRemovedByUser, 228
keyboard events, 150-152
mouse events
capturing, 155-156
drag-and-drop events, 154-155
explained, 152-153
MouseButtonEventArgs, 154
MouseEventArgs, 153-154
MouseWheelEventArgs, 154
transparent and null regions, 153
navigation events, 200-201
order of processing, 22
rendering, 607
routed events
About dialog example, 144-146
adding to user controls, 727-728
attached events, 147-149
consolidating routed event handlers, 149-150
defined, 141
explained, 141-142
implementation, 142-143
RoutedEventArgs class, 144
routing strategies, 143-144
stopping, 147
SelectedDatesChanged, 340
SelectionChanged, 261, 266
stylus events, 156-158
toast notification events, 778-779
touch events
basic touch events, 159-162
explained, 158
manipulation events, 162-170
triggers, 67
WPF 4.5 enhancements, 15
wrappers, 142

EventTriggers, 619-620

evolution of WPF. *See* releases of WPF

ExceptionValidationRule object, 407

Execute method, 171

executing commands with input gestures, 175

Expander class, 253-254

Expansion property (ManipulationDelta class), 163

explicit sizes, avoiding, 79

explicit versus implicit runs, 314

ExponentialEase function, 638

Expression Blend, 12

expressions, 71

ExtendGlassFrame method, 235

extensibility mechanisms, attached properties as, 75

extensibility of XAML, 35

Extensible Application Markup Language. *See* XAML

F

factoring XAML, 355-356

FactoryMethod keyword, 47-50

Failed event, 779

FanCanvas, 764-768

FieldModifier keyword, 50

FileInputBox control

- behavior, 721-723
- dependency properties, 724-727
- protecting from accidental usage, 723-724
- routed events, 727-728
- user interface, 719-721

files. *See also specific files*

- code-behind files, 40
- MainWindow.xaml.cs, 160-161, 168-169
- raw project files, opening in Visual Studio, 348
- VisualStudioLikePanes.xaml, 134-136
- VisualStudioLikePanes.xaml.cs, 136-140

FillBehavior property (animation classes), 619

FillRule property (PathGeometry class), 480-481

Filter property (ICollectionView), 391

filtering, 391

finding type converters, 28

FindResource method, 357

FirstDayOfWeek property (Calendar control), 339

Flat line cap (Pen), 488

flow documents

- annotations, 331-334
- Blocks
  - AnchoredBlock class, 326-327
  - BlockUIContainer, 321
  - List, 320
  - Paragraph, 320
  - sample code listing, 321-324
  - Section, 320
  - Table, 320
- creating, 318-319
- defined, 318
- displaying, 329-331
- Inlines
  - AnchoredBlock, 326-327
  - defined, 324-325
  - InlineUIContainer, 329
  - LineBreak, 327
  - Span, 325-326

FlowDirection property (FrameworkElements class), 85-86

FlowDocument element, 318

FlowDocumentPageViewer control, 329

FlowDocumentReader, 331-333

FlowDocumentReader control, 329

FlowDocumentScrollViewer control, 329

FontSizeConverter type converter, 28

Form1.cs file, 700-703

FormatConvertedBitmap class, 310

formatting strings, 374-376

Frame class, 194-196, 251-252

frame-based animation, 607

FrameworkContentElement class, 57, 62, 318

FrameworkElement class, 62

- explained, 57
- Triggers property, 67

FrameworkElements class

- ActualHeight property, 80
- ActualWidth property, 80
- DesiredSize property, 79
- FlowDirection property, 85-86
- Height property, 78-80
- HorizontalAlignment property, 83-84
- HorizontalContentAlignment property, 84-86
- LayoutTransform property, 86
- Margin property, 80-82
- Padding property, 80-82

RenderSize property, 79
RenderTransform property, 86
VerticalAlignment property, 83-85
Visibility property, 82-83
Width property, 78-80
FrameworkPropertyMetadata, 727
Freezable class, 56
freezing columns, 277
From property (animation classes), 612-614
FromArgb method, 703
FrozenColumnCount property (DataGrid), 277
full-trust XAML Browser applications, 210
functions. *See specific functions*

G

gadget-style applications, 205-206
galleries, RibbonGallery control, 297-298
GDI (graphics device interface), 8
hardware acceleration and, 11
GDI+, 8
generated source code, 42
generic dictionaries, 467, 737-738
generics support (XAML2009), 45
geometries
aggregate geometries, 481
Bézier curves, 478
CombinedGeometry class, 484-485
defined, 477
EllipseGeometry class, 477
Geometry3D class, 576
GeometryGroup class, 482-484
LineGeometry class, 477
MeshGeometry3D class, 576-577
Normals property, 579-581
Positions property, 577
TextureCoordinates property, 581
TriangleIndices property, 578-579
PathGeometry class
ArcSegment, 478
BezierSegment, 478
example, 478-480
explained, 477
FillRule property, 480-481
LineSegment, 478
PolyBezierSegment, 478
PolyLineSegment, 478
PolyQuadraticBezierSegment, 478
QuadraticBezierSegment, 478
RectangleGeometry class, 477
representing as strings, 485-487
StreamGeometry class, 481
Geometry3D class, 576
GeometryCombineMode enumeration, 484
GeometryDrawing class, 474-475
GeometryGroup class, 482-484
GeometryModel3D
defined, 561
explained, 569
Geometry3D class, 576
Materials
AmbientMaterial, 573
combining, 576
DiffuseMaterial, 570-573
EmissiveMaterial, 574
explained, 569
SpecularMaterial, 575-576
MeshGeometry3D class, 576-577
Normals property, 579-581
Positions property, 577
TextureCoordinates property, 581
TriangleIndices property, 578-579
GestureOnly value (InkCanvasEditingMode), 318
GetCommandLineArgs method, 184
GetErrors method, 408
GetExceptionForHR method, 47
GetGeometry method, 477

GetHbitmap function, 704
GetInstalledVoices method, 662
GetIntermediateTouchPoints method, 159
GetObject value (NodeType property), 787
GetPosition method, 153-155, 157
GetTouchPoint method, 159
GetValueSource method, 70
GetVisualChild method, 495-496
GlyphRunDrawing class, 474
GradientOrigin property (RadialGradientBrush), 517
gradients
GradientStop objects, 513
LinearGradientBrush class, 513-516
RadialGradientBrush class, 517-518
transparent colors, 518
GradientSpreadMethod enumeration, 515
GradientStop objects, 513
GrammarBuilder class, 668-669
grammars
GrammarBuilder class, 668-669
Speech Recognition Grammar Specification (SRGS), 667-668
graphics device interface (GDI), 8
graphics hardware, 9
graphics. *See* 2D graphics; 3D graphics
Grid
cell properties, 111
compared to other panels, 119
explained, 108
interaction with child layout properties, 120
interactive sizing with GridSplitter, 115-116
mimicking Canvas with, 119
mimicking DockPanel with, 119
mimicking StackPanel with, 119
sharing row/column sizes, 117-119
ShowGridLines property, 112
sizing rows/columns, 112-115
absolute sizing, 113
autosizing, 113
GridLength structures, 114-115
percentage sizing, 114
proportional sizing, 113
start page with Grid, 108-112
GridLength structures, 114-115
GridLengthConverter, 114
GridSplitter class, 115-116
GridView control, 270-271
GridViewColumn object, 270
GroupBox class, 253
GroupDescriptions property (ICollectionView), 387
grouping, 387-390
GroupName property (RadioButton class), 247
GroupSizeDefinitions property (RibbonGroup), 286-288
GroupSizeReductionOrder property (RibbonTab), 285

H

Handled property (RoutedEventArgs class), 144
HandleRef, 680
hardware acceleration
explained, 10
GDIand, 11
HasContent property (ContentControl class), 242
HasItems property (ItemsControl class), 256
Header property (ToolBar), 306
HeaderedItemsControl class, 299
headers
containers with headers
Expander class, 253-254
GroupBox class, 253
headered items controls, 279, 299
Height property (FrameworkElements class), 78-80
Help command, 173-174
Hidden value (Visibility enumeration), 82
HierarchicalDataTemplate, 379, 398-399

hijacking dependency properties, 441
Hillberg, Mike, 383
hit testing
3D hit testing, 590-591
input hit testing
explained, 497
InputHitTest method, 511
visual hit testing
callback methods, 503
explained, 497
simple hit testing, 497-498
with multiple Visuals, 498-501
with overlapping Visuals, 501-503
HitTest method, 500-503
HitTestCore method, 503
HitTestFilterCallback delegate, 502
HitTestResultCallback delegates, 501
HorizontalAlignment property
Canvas, 99
DockPanel, 108
FrameworkElements, 83-84
Grid, 120
StackPanel, 101
WrapPanel, 104
HorizontalContentAlignment property (FrameworkElements class), 84-85
HostingWin32.cpp file, 681
HostingWPF.cpp file, 689-693
house drawing, 536-537
2D drawing, 536
3D drawing, 537-538
Transform3Ds, 553-554
HwndHost class, 681
HwndSource class, 688-691
HwndSource variable, 693-694
hyperlinks, 197-198

I

ICC (International Color Consortium), 513
ICollectionViewLiveShaping interface, 395
ICommand interface, 171
Icon property (MenuItem class), 299
IconResourceIndex property (JumpTask), 222
IconResourcePath property (JumpTask), 222
ICustomTypeDescriptor interface, 366
IDs, AppUserModelIDs, 772-773
IEasingFunction interface, 638
IList interface, 32
Image control, 309-311
ImageBrush class, 522-523
ImageDrawing class, 474-476
images. *See* 2D graphics; 3D graphics
ImageSource class, 310
ImageSourceConverter type converter, 309
ImeProcessedKey property (KeyEventArgs event), 150
immediate mode systems, 12, 473
implicit .NET namespaces, 23
implicit styles, creating, 421-422
implicit versus explicit runs, 314
InAir property (StylusDevice class), 157
Indeterminate ProgressState, 230
inertia, enabling, 165-170
Ingebretsen, Robby, 19
inheritance
class hierarchy, 55-57
property value inheritance, 67-69
styles, 418
InitializeComponent method, 42, 44, 180
InitialShowDelay property (ToolTip class), 250
InkAndGesture value (InkCanvasEditingMode), 318
InkCanvas control, 316-318
Inline elements
AnchoredBlock, 326-327
defined, 324-325
InlineUIContainer, 329

LineBreak, 327
Span, 325-326
Inlines property (TextBlock control), 314
InlineUIContainer class, 329
InnerConeAngle property (PointLights), 566
INotifyDataErrorInfo interface, 408-409
input gestures, 175
input hit testing
explained, 497
InputHitTest method, 511
InputGestureText property (MenuItem class), 300
InputHitTest method, 511
inspecting WPF elements, 12
instantiating objects
with factory methods, 47-48
with non-default constructors, 47
Int16 keyword, 50
Int32 keyword, 50
Int64 keyword, 50
integration of WPF, 9
IntelliSense, 53
intensity of lights, 563
interfaces. ***See specific interfaces***
International Color Consortium (ICC), 513
interoperability (WPF)
ActiveX content, 710-714
C++/CLI, 677
DirectX, 13, 704-710
explained, 671-673
overlapping content, 673
Win32 controls
explained, 673-674
HwndSource class, 688-691
keyboard navigation, 683-687
launching modal dialogs, 688, 695
layout, 692-695
Webcam control, 674-683
Windows Forms controls
converting between two representatives, 703-704
ElementHost class, 700-702
explained, 695-696
launching modal dialogs, 699, 704
PropertyGrid, 696-699
InvalidItem value (JumpItemRejectionReason enumeration), 228
Inverted property (StylusDevice class), 157
IsAdditive property (animation classes), 619
IsAsync property (Binding object), 401
IsCheckable property (MenuItem class), 300
IsChecked property (ToggleButton class), 245
IsCumulative property (animation classes), 619
IsDefault property (Button class), 63-64, 244
IsDefaulted property (Button class), 244
IsDown property (KeyEventArgs event), 151
IsDropDownOpen property (Ribbon), 280
IsEditable property (ComboBox), 262
IsFrontBufferAvailableChanged event handler, 708
IsGrouping property (ItemsControl class), 256
IsHostedInRibbonWindow property (Ribbon), 281
IsIndeterminate property (ProgressBar control), 335
IsKeyboardFocused property (UIElements class), 152
IsKeyDown method, 152
IsMinimized property (Ribbon), 280
IsMouseDirectlyOver property (UIElements class), 153
IsNetworkDeployed method, 213
isolated storage, 191-192
IsolatedStorage namespace, 192
IsolatedStorageFile class, 192
IsolatedStorageFileStream class, 192
IsPressed property (ButtonBase class), 243
IsReadOnly property (ComboBox), 262
IsRepeat property (KeyEventArgs event), 151
IsSelected property (Selector class), 261
IsSelectionActive property (Selector class), 261
IsSynchronizedWithCurrentItem method, 372
IsSynchronizedWithCurrentItem property (Selector), 371
IsTextSearchCaseSensitive property (ItemsControl class), 265

IsTextSearchEnabled property (ItemsControl class), 265
IsThreeState property (ToggleButton class), 245
IsToggled property (KeyEventArgs event), 151
IsUp property (KeyEventArgs event), 151
ItemHeight property (WrapPanel), 102
items controls
ComboBox
ComboBoxItem objects, 266-267
customizing selection box, 262-265
events, 262
explained, 262
IsEditable property, 262
IsReadOnly property, 262
SelectionChanged event, 266
ContextMenu, 301-302
DataGrid
auto-generated columns, 274-275
CanUserAddRows property, 278
CanUserDeleteRows property, 278
clipboard interaction, 276
ClipboardCopyMode property, 276
column types, 273-274
displaying row details, 276-277
editing data, 277-278
EnableColumnVirtualization property, 276
EnableRowVirtualization property, 276
example, 272-273
freezing columns, 277
FrozenColumnCount property, 277
RowDetailsVisibilityMode property, 277
selecting rows/cells, 275
SelectionMode property, 275
SelectionUnit property, 275
virtualization, 276
GridView, 270-271
ItemsControl class, 255-256
AlternationCount property, 256
AlternationIndex property, 256
DisplayMemberPath property, 256-257
HasItems property, 256
IsGrouping property, 256
IsTextSearchCaseSensitive property, 265
IsTextSearchEnabled property, 265
Items property, 255
ItemsPanel property, 256-260
ItemsSource property, 256
ListBox
automation IDs, 269
example, 267-268
scrolling, 269
SelectionMode property, 268
sorting items in, 269
support for multiple selections, 268
ListView, 270-271
Menu, 298-301
Ribbon
application menu, 291-292
contextual tabs, 294-295
example, 278-281
galleries, 297-298
IsDropDownOpen property, 280
IsHostedInRibbonWindow property, 281
IsMinimized property, 280
key tips, 289-290
minimizing, 280
Quick Access toolbar, 292-294
resizing, 284-289
ribbon controls, 281-284, 288-289
RibbonGroup, 279, 286-288
RibbonTab, 279, 285-286
RibbonWindow, 280-281
ScreenTips, 295-296
WindowIconVisibility property, 281
scrolling behavior, controlling, 260-261
Selector class, 261
StatusBar, 307-308
TabControl, 271-272

ToolBar, 304-306
TreeView, 302-304
items panels, 258
Items property (ItemsControl class), 255, 371
ItemsCollection object, 269
ItemsControl class, 255-256
AlternationCount property, 256
AlternationIndex property, 256
DisplayMemberPath property, 256-257
HasItems property, 256
IsGrouping property, 256
IsTextSearchCaseSensitive property, 265
IsTextSearchEnabled property, 265
Items property, 255
ItemsPanel property, 256-260
ItemsSource property, 256
scrolling behavior, controlling, 260-261
ItemsPanel property (ItemsControl class), 256-260
ItemsSource collection, 277
ItemsSource property (ItemsControl class), 256, 371
ItemWidth property (WrapPanel), 102
IValueConverter interface, 380-382
IXamlLineInfo interface, 788

J

journal, 198-200
JournalOwnership property (Frame class), 198-199
Jump Lists
associating with applications, 218
explained, 218
JumpPaths
adding, 226-227
explained, 225
recent and frequent JumpPaths, 227-228
responding to rejected or removed items, 228
JumpTasks
customizing behavior of, 221-224
example, 218-219
explained, 218
and Visual Studio debugger, 220
JumpItemRejectionReason enumeration, 228
JumpItemsRejected event, 228
JumpItemsRemovedByUser event, 228
JumpList class. *See* Jump Lists
JumpPaths
adding, 226-227
explained, 225
recent and frequent JumpPaths, 227-228
responding to rejected or removed items, 228
JumpTasks
customizing behavior of, 221-224
example, 218-219
explained, 218

K

Kaxaml, 18-19
Key enumeration, 150-152
Key keyword, 50, 352
Key property (KeyEventArgs event), 150
key tips, 289-290
keyboard events, 150-152
keyboard navigation
customizing, 306
supporting in Win32 controls, 683-684
access keys, 687
tabbing in/out of Win32 content, 684-686
keyboard support for CheckBox control, 246
KeyboardDevice property (KeyEventArgs event), 151
KeyboardNavigation class, 306
KeyDown event, 150
KeyEventArgs event, 150-151
keyframe animation
discrete keyframes, 632-634
easing keyframes, 634

explained, 628
linear keyframes, 629-631
spline keyframes, 631-632
keyless resources, 422-423
KeyStates property
KeyEventArgs event, 151
QueryContinueDragEventArgs class, 155
KeyTip property (ribbon controls), 289-290
KeyUp event, 150
keywords. *See specific keywords*

L

Label class, 248
Label property (ribbon controls), 283
Language Integrated Query (LINQ), 396
LargeImageSource property (ribbon controls), 283
LastChildFill property (DockPanel), 105
launching modal dialogs
from Win32 applications, 695
from Windows Forms applications, 704
from WPF applications, 688, 699
layout
content overflow, handling
clipping, 122-124
explained, 122
scaling, 126-130
scrolling, 124-126
custom panels
communication between parents and children, 748-751
explained, 747-748
FanCanvas, 764-768
OverlapPanel, 759-764
SimpleCanvas, 751-756
SimpleStackPanel, 756-759
embedding WPF controls in Win32 applications, 692-695
explained, 77-78
panels, 77
Canvas, 98-100
DockPanel, 105-108
explained, 97-98
Grid. *See* Grid
SelectiveScrollingGrid, 121-122
StackPanel, 100-101
TabPanel, 120
ToolBarOverflowPanel, 121
ToolBarPanel, 121
ToolBarTray, 121
UniformGrid, 121
WrapPanel, 102-105
positioning elements
content alignment, 84-85
explained, 83
flow direction, 85-86
horizontal and vertical alignment, 83-84
stretch alignment, 84
sizing elements
explained, 78
explicit sizes, avoiding, 79
height and width, 78-80
margin and padding, 80-82
visibility, 82-83
transforms
applying, 86-88
combining, 94
explained, 86
MatrixTransform, 93
RotateTransform, 88-90
ScaleTransform, 90-92
SkewTransform, 92
support for, 94-95
TranslateTransform, 92-93
Visual Studio-like panes, creating
sequential states of user interface, 130-134
VisualStudioLikePanes.xaml, 134-136
VisualStudioLikePanes.xaml.cs, 136-140

LayoutTransform property
Canvas panel, 99
DockPanel, 108
FrameworkElements, 86
Grid, 120
StackPanel, 101
WrapPanel, 105
Left property (Canvas), 98
LengthConverter type converter, 82
Light and Fluffy skin example, 463-464
Light objects
AmbientLight, 562, 567-568
defined, 561
DirectionalLight, 562-563
explained, 540, 561
intensity of, 563
PointLight, 562-564
SpotLight, 562-566
Line class, 507-508
linear interpolation, 610-611
linear keyframes, 629-631
LinearAttenuation property (PointLights), 564
LinearGradientBrush class, 513-516
LineBreak class, 327
LineGeometry class, 477
LineJoin property (Pen class), 488
LineSegment class, 478
LINQ (Language Integrated Query), 396
List Blocks, 320
ListBox control
arranging items horizontally, 259
automation IDs, 269
example, 267-268
placing PlayingCards custom control into, 738
scrolling, 269
SelectionMode property, 268
sorting items in, 269
support for multiple selections, 268
lists, 32-33
Jump Lists
and Visual Studio debugger, 220
associating with applications, 218
explained, 218
JumpPaths, 225-228
JumpTasks, 218-224
List Blocks, 320
ListBox
arranging items horizontally, 259
automation IDs, 269
example, 267-268
placing PlayingCards custom control into, 738
scrolling, 269
SelectionMode property, 268
sorting items in, 269
support for multiple selections, 268
ListView, 270-271
ListView control, 270-271
live objects, writing to, 791-793
live shaping, 385, 395
Load method, 36-37, 794
LoadAsync method (XamlReader), 37
LoadComponent method, 43
loading XAML at runtime, 36-37
Lobo, Lester, 19
local values, clearing, 71
localization IDs, marking user interfaces with, 349
localizing binary resources, 348-349
LocBaml, creating satellite assembly with, 349
locking D3DImage, 709
logical AND relationships, 429-430
logical OR relationships, 429
logical resources
accessing directly, 358
consolidating color brushes with, 351-353
defining and applying in procedural code, 357-358
explained, 349-350

interaction with system resources, 358-359
resource lookup, 353
resources without sharing, 356
static versus dynamic resources, 353-355
logical trees, 57-62
LogicalChildren property, 62
LogicalTreeHelper class, 59
LookDirection property (Cameras), 542-546
lookup (resource), 353
loose XAML pages, 213-214

M

mage.exe command-line tool, 192
mageUI.exe graphical tool, 192
Main method, 181-183
MainWindow class, 179-180
MainWindow.xaml file, 706
MainWindow.xaml.cs file, 160-161, 168-169, 706-708
malicious skins, preventing, 464-465
managed code, mixing with unmanaged code, 678
manipulation events
adding inertia with, 165-170
enabling panning/rotating/zooming with, 164-165
explained, 162-163
ManipulationCompleted, 163
ManipulationDelta, 163
ManipulationStarted, 163
ManipulationStarting, 163
ManipulationBoundaryFeedback event, 167
ManipulationCompleted event, 163
ManipulationDelta event, 163-165
ManipulationDeltaEventArgs instance, 163
ManipulationInertiaStarting event, 165, 169
ManipulationStarted event, 163
ManipulationStarting event, 163
Margin property
Canvas panel, 99
DockPanel, 108
FrameworkElements, 80-82
Grid, 120
StackPanel, 101
WrapPanel, 104
markup compatibility, 791
markup extensions
explained, 28-31
parameters, 29
in procedural code, 31
Materials
AmbientMaterial, 573
combining, 576
DiffuseMaterial, 570-573
EmissiveMaterial, 574
explained, 569
SpecularMaterial, 575-576
MatrixCamera class, 551
MatrixTransform, 93
MatrixTransform3D class, 560
MDI (multiple-document interface), 185
MeasureOverride method, 748-750
MediaCommands class, 172
MediaElement class
playing audio, 654-656
playing video, 656-658
MediaPlayer class, 653-654
MediaTimeline class
playing audio, 654-656
playing video, 659-660
Members keyword, 49-50
Menu control, 298-301
MenuItem class, 299
menus
ContextMenu control, 301-302
Menu control, 298-301
MergedDictionaries property (ResourceDictionary class), 355

MeshGeometry3D class, 576-577
Normals property, 579-581
Positions property, 577
TextureCoordinates property, 581
TriangleIndices property, 578-579
methods. *See specific methods*
minimizing Ribbon, 280
missing styles, troubleshooting, 461
mnemonics, 687
modal dialogs
launching from Win32 applications, 695
launching from Windows Forms applications, 704
launching from WPF applications, 688, 699
Model3DGroup, defined, 561
Model3DGroup class, 582-584
Model3Ds
explained, 561
GeometryModel3D
defined, 561
explained, 569
Geometry3D class, 576
Materials, 569-576
MeshGeometry3D class, 576-581
Lights
AmbientLight, 562, 567-568
DirectionalLight, 562-563
explained, 561
intensity of, 563
PointLight, 562-564
SpotLight, 562-566
Model3DGroup, 561, 582-584
modeless dialogs, 178
ModelUIElement3D class, 586-588
ModelVisual3D class, 585-586
Modifiers property (KeyboardDevice), 151
Mouse class, 155
mouse events
capturing, 155-156
drag-and-drop events, 154-155
explained, 152-153
MouseButtonEventArgs, 154
MouseEventArgs, 153-154
MouseWheelEventArgs, 154
transparent and null regions, 153
MouseButtonEventArgs class, 154
MouseButtonState enumeration, 153
MouseEventArgs class, 153-154
MouseOverBackground property (ribbon controls), 283
MouseOverBorderBrush property (ribbon controls), 283
MouseWheelEventArgs class, 154
MultiBinding class, 410-411
multiple providers, support for
applying animations, 71
coercion, 71
determining base values, 69-70
evaluating, 71
explained, 69
validation, 72
multiple visuals, hit testing with, 498-501
multiple-document interface (MDI), 185
MultipleRange value (SelectionMode), 337
MultiPoint Mouse SDK, 158
multithreaded applications, 187
MyHwndHost class, 680-682

N

Name keyword, 38, 50
named elements, 434
named styles, 421-422
NamespaceDeclaration value (NodeType property), 788

namespaces
explained, 22-24
implicit .NET namespaces, 23
mapping, 22
naming elements, 38-39
Navigate method, 196-197
navigation
keyboard navigation, supporting in Win32 controls, 683-687
views, 391-392
XAMLBrowser applications, 210-211
navigation-based desktop applications
explained, 193-194
hyperlinks, 197-198
journal, 198-200
Navigate method, 196-197
navigation containers, 194-196
navigation events, 200-201
Page elements, 194-196
returning data from pages, 203-204
sending data to pages, 202-203
NavigationCommands class, 172
NavigationProgress event, 201
NavigationStopped event, 201
NavigationWindow class, 194-196
nearest-neighbor bitmap scaling, 310
.NET properties, binding to, 365-367
NodeType property (XAML), 787-788
None ProgressState, 230
None value
InkCanvasEditingMode, 318
NodeType property, 788
SelectionMode, 337
nonprinciple axis, scaling about, 557
NonZero fill (FillRule property), 480
NoRegisteredHandler value (JumpItemRejectionReason enumeration), 228
Normal ProgressState, 230
normals, 579
Normals property (MeshGeometry3D class), 579-581
notifications (toast)
audio, 778
canceling, 780
changing color of, 775
notification events, 778-779
prerequisites, 771-773
scheduled notifications, 779-780
sending, 774-775
templates, 775-778
NotificationsExtensions project, 777
Null keyword, 52
null regions and mouse events, 153

O

Object class, 55
object elements
attributes, 21
content property, 31-32
declaring, 21
dictionaries, 33-34
explained, 20-21
lists, 32-33
naming, 38-39
positioning
content alignment, 84-85
explained, 83
flow direction, 85-86
horizontal and vertical alignment, 83-84
stretch alignment, 84
processing child elements, 36
sizing
explained, 78
explicit sizes, avoiding, 79
height and width, 78-80
margin and padding, 80-82
visibility, 82-83

transforms
applying, 86-88
combining, 94
explained, 86
MatrixTransform, 93
RotateTransform, 88-90
ScaleTransform, 90-92
SkewTransform, 92
support for, 94-95
TranslateTransform, 92-93
values type-converted to object elements, 34
Object keyword, 51
ObjectDataProvider class, 400-402
objects
binding to, 367-368
instantiating via factory methods, 47-48
instantiating with non-default constructors, 47
live objects, writing to, 791-793
logical trees, 57-58
visual trees, 58-62
on-demand download, 212-213
OneTime binding, 403
OneWay binding, 403
OneWayToSource binding, 403-404
OnMnemonic method, 687
OnNoMoreTabStops method, 686
opacity masks, brushes as, 525-527
Opacity property (brushes), 525
OpacityMask property (brushes), 525-527
OpenGL, 8
opening project files in Visual Studio, 348
OR relationships (logical), 429
order of property and event processing, 22
Orientation property
ProgressBar control, 335
StackPanel, 100
WrapPanel, 102
OriginalSource property (RoutedEventArgs class), 144
OrthographicCamera class
blind spots, 543
compared to PerspectiveCamera class, 549-551
LookDirection property, 542-546
Position property, 541-542
UpDirection property, 546-548
Z-fighting, 543
OuterConeAngle property (PointLights), 566
OverlapPanel, 759-764
overlapping content, 673
overlapping visuals, hit testing with, 501-503
Overlay property (TaskbarItemInfo), 231
overlays, adding to taskbar items, 231
overriding
ArrangeOverride method, 750-751
MeasureOverride method, 748-750

P

packageURI, 347
Padding property (FrameworkElements class), 80-82
Page elements, 194-196
PageFunction class, 203-204
pages
loose XAML pages, 213-214
Page elements, 194-196
refreshing, 199
returning data from, 203-204
sending data to, 202-203
stopping loading, 199
panels, 77
Canvas, 98-100, 119
content overflow, handling
clipping, 122-124
explained, 122
scaling, 126-130
scrolling, 124-126
custom panels
communication between parents and children, 748-751

explained, 747-748
FanCanvas, 764-768
OverlapPanel, 759-764
SimpleCanvas, 751-756
SimpleStackPanel, 756-759
DockPanel
examples, 105-107
explained, 105
interaction with child layout properties, 107-108
mimicking with Grid, 119
properties, 105
explained, 97-98
Grid
cell properties, 111
compared to other panels, 119
explained, 108
interaction with child layout properties, 120
interactive sizing with GridSplitter, 115-116
mimicking Canvas with, 119
mimicking DockPanel with, 119
mimicking StackPanel with, 119
sharing row/column sizes, 117-119
ShowGridLines property, 112
sizing rows/columns, 112-115
start page with Grid, 108-112
SelectiveScrollingGrid, 121-122
StackPanel
explained, 100
interaction with child layout properties, 101
mimicking with Grid, 119
TabPanel, 120
ToolBarOverflowPanel, 121
ToolBarPanel, 121
ToolBarTray, 121
UniformGrid, 121
Visual Studio-like panes, creating
sequential states of user interface, 130-134
VisualStudioLikePanes.xaml, 134-136
VisualStudioLikePanes.xaml.cs, 136-140
WrapPanel
and right-to-left environments, 104
examples, 103-104
explained, 102
interaction with child layout properties, 104-105
properties, 102
panning
enabling with touch events, 164-165
with inertia, 166-167
Paragraph Blocks, 320
Parse method, 794
parsing XAML at runtime, 36-38
partial keyword, 40
partial-trust applications, 13
parts (control), 447-449
PasswordBox control, 316
Path class, 509-510
path-based animations, 635
PathGeometry class
ArcSegment, 478
BezierSegment, 478
example, 478-480
explained, 477
FillRule property, 480-481
LineSegment, 478
PolyBezierSegment, 478
PolyLineSegment, 478
PolyQuadraticBezierSegment, 478
QuadraticBezierSegment, 478
Paused ProgressState, 230
Pen class, 487-489
percentage sizing, 114
performance
cached composition
BitmapCache class, 531-533
BitmapCacheBrush class, 533
Viewport2DVisual3D support for, 589

improving rendering performance
BitmapCache class, 531-533
BitmapCacheBrush class, 533
RenderTargetBitmap class, 530-531
XAML, 53
WPF 3.5 enhancements, 14
WPF 4 enhancements, 15
WPF 4.5 enhancements, 16
persisting application state, 192
PerspectiveCamera class
blind spots, 543
compared to OrthographicCamera class, 549-551
LookDirection property, 542-546
Position property, 542
UpDirection property, 546-548
Z-fighting, 543
Petzold, Charles, 19
PinToStart() method, 773
PixelFormats enumeration, 311
pixels
device-independent pixels, 82
pixel boundaries, 15
pixel shaders, 529
Play method, 652
PlayingCard control
behavior
code-behind file, 730
final implementation, 733-735
initial implementation, 729-733
resources, 730-731
generic resources, 737-738
placing into ListBox, 738
user interface, 735-738
PointLight, 562-564
PolyBezierSegment class, 478
Polygon class, 509
Polyline class, 508
PolyLineSegment class, 478
PolyQuadraticBezierSegment class, 478
Position property (Cameras), 541-542
positioning elements
content alignment, 84-85
explained, 83
flow direction, 85-86
horizontal and vertical alignment, 83-84
stretch alignment, 84
Positions property (MeshGeometry3D class), 577
power easing functions, 635-636
PreferUnconvertedDictionaryKeys property, 46
PressureFactor property (StylusPoint object), 157
PreviewKeyDown event, 150
PreviewKeyUp event, 150
printing logical/visual trees, 60-61
PrintLogicalTree method, 61
PrintVisualTree method, 60
PriorityBinding class, 411-412
procedural code
accessing binary resources from, 347-348
animation classes
AutoReverse property, 616
BeginTime property, 614-615
DoubleAnimation, 609-610
Duration property, 612
EasingFunction property, 618
explained, 606-608
FillBehavior property, 619
From property, 612-614
IsAdditive property, 619
IsCumulative property, 619
lack of generics, 608-609
linear interpolation, 610-611
RepeatBehavior property, 616-617
reusing animations, 611
SpeedRatio property, 615
To property, 612-614
Binding object in, 361-363

compared to XAML, 20
defining and applying resources in, 357-358
embedding PropertyGrid with, 696-698
frame-based animation, 607
markup extensions in, 31
mixing XAML with
BAML (Binary Application Markup Language), 41-44
CAML (Compiled Application Markup Language), 42
compiling XAML, 39-41
generated source code, 42
loading and parsing XAML at runtime, 36-38
naming XAML elements, 38-39
procedural code inside XAML, 43
skins, 462
timer-based animation, 606
type converters in, 27
procedural code, timer-based animation, 607
ProgressBar control, 335
adding to taskbars, 230
pie chart control template, 442-444, 453-455
ProgressState property (TaskbarItemInfo), 230
ProgressValue property (TaskbarItemInfo), 230
project files, opening in Visual Studio, 348
PromptBuilder class, 663-665
properties. ***See also specific properties***
attributes, 21
dependency properties
attached properties, 72-75
attached property providers, 74-75
change notification, 65-67
explained, 62-63
implementation, 63-65
property value inheritance, 67-69
support for multiple providers, 69-72
.NET properties, binding to, 365-367
order of processing, 22
property triggers, 65-67
property wrappers, 64-65
Properties collection, 185
property elements, 25-26
Property keyword, 49-51
property paths, 257
property triggers, 65-67, 424-427, 626-627
property value inheritance, 67-69
property wrappers, 64-65
PropertyGrid
embedding with procedural code, 696-698
embedding with XAML, 698-699
PropertyGroupDescription class, 389
proportional sizing, 113
protecting controls from accidental usage, 723-724
pure-XAML Twitter client, 412-414

Q

QuadraticAttenuation property (PointLights), 564
QuadraticBezierSegment class, 478
QuaternionRotation3D class, 557
QueryContinueDragEventArgs class, 155
Quick Access toolbar, 292-294

R

RadialGradientBrush class, 517-518
RadioButton class, 246-248
RadiusX property
RadialGradientBrush, 517
Rectangle, 505
RadiusY property
RadialGradientBrush, 517
Rectangle, 505

range controls
explained, 334
ProgressBar, 335
Slider, 335-336
Range property (PointLights), 564
raw project files, opening in Visual Studio, 348
readers (XAML)
explained, 783-785
markup compatibility, 791
node loops, 786-787
NodeType property, 787-788
sample XAML content, 789
XAML node stream, 789-791
XamlServices class, 794-797
recent and frequent JumpPaths, 227-228
Rectangle class, 505-506
RectangleGeometry class, 477
Reference keyword, 52
Refresh method, 199
refreshing pages, 199
Register method, 64
rejected items, reponding to, 228
RelativeSource property (Binding object), 365
releases of WPF
WPF 3.0, 12
WPF 3.5, 12-14
WPF 3.5 SP1, 13-14
WPF 4, 14-15
WPF 4.5, 12, 15-16
WPF Toolkit, 12
removed items, reponding to, 228
RemovedByUser value (JumpItemRejectionReason enumeration), 228
RemoveFromSchedule method, 780
RemoveHandler method, 142-143
RenderAtScale property (BitmapCache class), 531
rendering
controlling
data templates, 376-379
explained, 373-374
string formatting, 374-376
value converters, 380-384
custom rendering, 497
performance, improving
BitmapCache class, 531-533
BitmapCacheBrush class, 533
RenderTargetBitmap class, 530-531
text, 15, 312
Rendering event, 607
RenderSize property (FrameworkElements class), 79
RenderTargetBitmap class, 530-531
RenderTransform property (FrameworkElements class), 86
RenderTransformOrigin property (UIElement class), 87
RepeatBehavior property (animation classes), 616-617
RepeatButton class, 245
ResizeBehavior property (GridSplitter), 116
ResizeDirection property (GridSplitter), 116
resizing Ribbon
resizing behavior, 284-285
ribbon controls, 288-289
RibbonGroup, 286-288
RibbonTab, 285-286
resolution independence, 10
Resource build action, 342-343
ResourceDictionary, 33-34, 355
ResourceDictionaryLocation parameter, 467
resources
binary resources
accessing, 343-348
defining, 342-343
explained, 341
localizing, 348-349
defined, 341
keyless resources, 422-423
logical resources
accessing directly, 358
consolidating color brushes with, 351-353
defining and applying in procedural code, 357-358
explained, 349-350

interaction with system resources, 358-359
resource lookup, 353
resources without sharing, 356
static versus dynamic resources, 353-355
for PlayingCard custom control, 730-731
Resources property, 350
responding to rejected or removed items, 228
restoring application state, 192
restricting style usage, 420-421
retained mode systems, 12
retained-mode graphics systems, 473-474
returning data from pages, 203-204
reusing animations, 611
Ribbon control, 15
application menu, 291-292
contextual tabs, 294-295
example, 278-281
galleries, 297-298
IsDropDownOpen property, 280
IsHostedInRibbonWindow property, 281
IsMinimized property, 280
key tips, 289-290
minimizing, 280
Quick Access toolbar, 292-294
resizing
resizing behavior, 284-285
ribbon controls, 288-289
RibbonGroup, 286-288
RibbonTab, 285-286
ribbon controls
examples, 282-283
overview, 281
properties, 283-284
resizing, 288-289
RibbonGroup, 279, 286-288
RibbonTab, 279, 285-286
RibbonWindow, 280-281
ScreenTips, 295-296
RibbonApplicationMenu class, 291-292
RibbonButton, 281-283
RibbonCheckBox, 281
RibbonComboBox, 282
RibbonGallery control, 297-298
RibbonGroup, 279, 286-288
RibbonQuickAccessToolBar control, 292-294
RibbonRadioButton, 281
RibbonSeparator, 282
RibbonSplitButton, 282
RibbonTab, 279, 285-286
RibbonTextBox, 282
RibbonToggleButton, 282
RibbonToolTip control, 295-296
RibbonTwoLineText, 282
RibbonWindow, 280-281
RichTextBox control, 316
Right property (Canvas), 98
right-hand rule, 541, 578
right-handed coordinate systems, 541-542
RotateTransform, 88-90
RotateTransform3D, 557-560
rotation
enabling with touch events, 164-165
with inertia, 166-167
RotateTransform3D class, 557-560
Rotation property (ManipulationDelta class), 163
routed events
About dialog example, 144-146
adding to user controls, 727-728
attached events, 147-149
consolidating routed event handlers, 149-150
defined, 141
explained, 141-142
implementation, 142-143
RoutedEventArgs class, 144
routing strategies, 143-144
stopping, 147

RoutedEvent property (RoutedEventArgs class), 144
RoutedEventArgs class, 144
RoutedUICommand objects, 172
routing strategies, 143-144
RoutingStrategy enumeration, 143
RowDetailsVisibilityMode property (DataGrid), 277
rows
DataGrid
displaying row details, 276-277
selecting, 275
Grid
sharing row/column sizes, 117-119
sizing, 112-116
Run method, 182-183
runtime, loading and parsing XAML at, 36-38

S

satellite assemblies, creating with LocBaml, 349
Save method, 795
Scale property (ManipulationDelta class), 163
ScaleTransform, 90-92, 127
ScaleTransform3D class, 555-557
ScaleX property (RotateTransform class), 90
ScaleY property (RotateTransform class), 90
scaling, 126-130
about nonprinciple axis, 557
nearest-neighbor bitmap scaling, 310
ScaleTransform3D class, 555-557
ScheduledToastNotification object, 779-780
scheduling toast notifications, 779-780
scope of typed styles, 421
ScreenTips with Ribbon control, 295-296
scRGB color space, 512
ScrollBars, 125-126
scrolling, 124-126
controlling in items controls, 260-261
ListBox control, 269
ScrollViewer control, 124-126
Section Blocks, 320
security in XAMLBrowser applications, 211
Select value (InkCanvasEditingMode), 318
SelectedDatesChanged event, 340
SelectedIndex property (Selector class), 261
SelectedItem property (Selector class), 261
SelectedValue property (Selector class), 261
selecting rows/cells, 275
selection boxes (ComboBox control), customizing, 262-265
SelectionChanged event, 261, 266
SelectionMode property
Calendar control, 337
DataGrid, 275
ListBox, 268
SelectionUnit property (DataGrid), 275
SelectiveScrollingGrid, 121-122
Selector class, 261
selectors (data template), 380
SelectVoice method, 662
SelectVoiceByHints method, 662
sending
data to pages, 202-203
toast notifications, 774-775
SendToast() method, 774, 777
Separator control, 299
SetBinding method, 363
SetCurrentValue method, 72
SetOutputToDefaultAudioDevice method, 663
SetOutputToWaveFile method, 663
SetResourceReference method, 357
Setters, 419-420
Settings class, 192
ShaderEffect, 529
Shapes
clip art based on Shapes, 510-511
Ellipse class, 506
explained, 503-504
how they work, 507

Line class, 507-508
overuse of, 505
Path class, 509-510
Polygon class, 509
Polyline class, 508
Rectangle class, 505-506
Shared keyword, 51, 356
sharing
data source with DataContext, 372-373
Grid row/column sizes, 117-119
resources without sharing, 356
styles, 418-420
shortcuts, AppUserModelIDs, 772-773
ShowDialog method, 190-191
ShowDuration property (ToolTip class), 250
ShowFrequentCategory property (JumpList class), 227
ShowGridLines property (Grid), 112
ShowOnDisabled property
ContextMenuService class, 302
ToolTipService class, 251
ShowRecentCategory property (JumpList class), 227
ShutdownMode enumeration, 184
Silicon Graphics OpenGL, 8
SimpleCanvas, 751-756
SimpleQuadraticEase class, 639
SimpleStackPanel, 756-759
SineEase function, 638
Single keyword, 51
single-instance applications, 186
single-threaded apartment (STA), 181
SingleDate value (SelectionMode), 337
SingleRange value (SelectionMode), 337
sizing. *See also* resizing Ribbon
elements
explained, 78
explicit sizes, avoiding, 79
height and width, 78-80
margin and padding, 80-82
visibility, 82-83
Grid rows/columns, 112-115
absolute sizing, 113
autosizing, 113
GridLength structures, 114-115
interactive sizing with GridSplitter, 115-116
percentage sizing, 114
proportional sizing, 113
sharing row/column sizes, 117-119
SkewTransform, 92
skins
defined, 415
examples, 459-461
explained, 458-462
Light and Fluffy skin example, 463-464
malicious skins, preventing, 464-465
missing styles, troubleshooting, 461
procedural code, 462
Skip method, 793
Slider control, 335-336
SmallImageSource property (ribbon controls), 283
snapshots of individual video frames, taking, 658
SnapsToDevicePixels property, 15, 532
Snoop, 12
snooze interval (toast notifications), 780
SolidColorBrush class, 512
SortDescription class, 394
SortDescriptions collection, 386
SortDescriptions property
ICollectionView, 385
ItemsCollection, 269
sorting, 269, 385-387
SoundPlayer class, 652
SoundPlayerAction class, 652-653
Source property
MediaElement class, 654
RoutedEventArgs class, 144
SourceName property (Trigger class), 433

spaces in geometry strings, 487
spans, 325-326
SpeakAsync method, 662
SpeakAsyncCancelAll method, 662
SpecularMaterial class, 575-576
speech recognition
converting spoken words into text, 665-667
specifying grammar with GrammarBuilder, 668-669
specifying grammar with SRGS, 667-668
Speech Recognition Grammar Specification (SRGS), 667-668
speech synthesis
explained, 662
GetInstalledVoices method, 662
PromptBuilder class, 663-665
SelectVoice method, 662
SelectVoiceByHints method, 662
SetOutputToWaveFile method, 663
SpeakAsync method, 662
Speech Synthesis Markup Language (SSML), 663-665
SpeechSynthesizer, 662
Speech Synthesis Markup Language (SSML), 663-665
SpeechRecognitionEngine class, 667
SpeechSynthesizer, 662
SpeedRatio property (animation classes), 615
spell checking in TextBoxes, 315
Spinning Prize Wheel, 168-169
splash screens, 187-188
spline keyframes, 631-632
SpotLight, 562-566
SpreadMethod property (LinearGradientBrush), 515
Square line cap (Pen), 488
sRGB color space, 512
SRGS (Speech Recognition Grammar Specification), 667-668
SSML (Speech Synthesis Markup Language), 663-665
STA (single-threaded apartment), 181
StackPanel, 101. ***See also*** **SimpleStackPanel**
explained, 100
interaction with child layout properties, 101
with Menu control, 300
mimicking with Grid, 119
setting font properties on, 72-73
standard desktop applications
Application class
creating applications without, 186
events, 184
explained, 181-182
Properties collection, 185
Run method, 182-183
Windows collection, 184
application state, 192
ClickOnce, 192-193
common dialogs, 188-189
custom dialogs, 189-190
explained, 177-178
multiple-document interface (MDI), 185
multithreaded applications, 187
retrieving command-line arguments in, 184
single-instance applications, 186
splash screens, 187-188
Window class, 178-180
Windows Installer, 192
start pages, building with Grid, 108-112
Start screen shortcuts, 772-773
starting animations from property triggers, 626-627
StartLineCap property (Pen class), 487
StartMember value (NodeType property), 787
StartObject value (NodeType property), 787
StartPoint property (LinearGradientBrush), 514
StartupUri property (Application class), 182-183
state groups, 450
states
control states, 449-455, 741-745
persisting and restoring, 192

state groups, 450
visual states
respecting with triggers, 442-446
respecting with VSM (Visual State Manager), 447-455
STAThreadAttribute, 691
Static keyword, 52
static versus dynamic resources, 353-355
StaticResource markup extension, 353-355
StatusBar control, 307-308
StopLoading method, 199
stopping
page loading, 199
routed events, 147
Storyboards
EventTriggers containing Storyboards, 619-620
TargetName property, 623-624
TargetProperty property, 620-623
as Timelines, 627-628
StreamGeometry class, 481
Stretch alignment, 84
Stretch enumeration, 127
Stretch property
DrawingBrush class, 519
MediaElement class, 656
StretchDirection enumeration, 127
StretchDirection property (MediaElement class), 656
String keyword, 51
StringFormat property (Binding object), 374
strings
formatting, 374-376
representing geometries as, 485-487
Stroke objects, 317
Style class. *See* styles
styles
consolidating property assignments in, 417
default styles, 70
defined, 415
explained, 416-418
implicit styles, creating, 421-422
inheritance, 418
keyless resources, 422-423
missing styles, troubleshooting, 461
mixing with control templates, 456-457
named styles, 421-422
per-theme styles and templates, 466-469
restricting usage of, 420-421
Setter behavior, 419-420
sharing, 418-420
theme styles, 70
triggers
conflicting triggers, 429
data triggers, 427-428
explained, 423-424
expressing logic with, 428-430
property triggers, 424-427
respecting visual states with, 442-446
typed styles, 421-422
stylus events, 156-158
StylusButtonEventArgs instance, 158
StylusButtons property (StylusDevice class), 157
StylusDevice class, 156-157
StylusDownEventArgs instance, 158
StylusEventArgs class, 158
StylusPoint objects, 157
StylusSystemGestureEventArgs instance, 158
Subclass keyword, 51
Surface Toolkit for Windows Touch, 170
SynchronousMode keyword, 51
system resources, interaction with logical resources, 358-359
system-defined background, setting, 359
SystemKey property (KeyEventArgs event), 150
SystemSounds class, 652

T

TabControl control, 271-272
TabInto method, 684
Table Blocks, 320
TabletDevice property (StylusDevice class), 157
TabPanel, 120
TargetName property (Storyboards), 623-624
TargetNullValue property (Binding object), 364
TargetProperty property (Storyboards), 620-623
TargetType property
ControlTemplate class, 434-435
Style class, 420-421
taskbar, customizing
explained, 230
taskbar item overlays, 231
taskbar item progress bars, 230
thumb buttons, 232-233
thumbnail content, 231
TaskDialogs, 236-239
TemplateBindingExtension class, 435-437
templated parent properties, respecting, 435-441
Content property (ContentControl class), 435-437
hijacking existing properties for new purposes, 441
other properties, 440
templates
control templates
editing, 457-458
mixing with styles, 456-457
named elements, 434
resuability of, 438-440
simple control template, 431-432
target type, restricting, 434-435
templated parent properties, respecting, 435-441
other properties, 438-439
triggers, 432-434
visual states, respecting with triggers, 442-446
visual states, respecting with VSM (Visual State Manager), 447-455
DataTemplates, 376-379, 398-399
defined, 415
explained, 430-431
per-theme styles and templates, 466-469
template selectors, 380
toast notification templates, 775-778
Windows themes, 470
temporarily canceling data binding, 384
testing
3D hit testing, 590-591
input hit testing
explained, 497
InputHitTest method, 511
visual hit testing
callback methods, 503
explained, 497
with multiple Visuals, 498-501
with overlapping Visuals, 501-503
simple hit testing, 497-498
text
converting spoken words into, 665-667
InkCanvas control, 316-318
PasswordBox control, 316
rendering, 15
RichTextBox control, 316
TextBlock control
explained, 313-314
explicit versus implicit runs, 314
properties, 313
support for multiple lines of text, 315
whitespace, 314
TextBox control, 315
TextOptions class, 312
WPF 3.5 enhancements, 13
WPF 4 enhancements, 15
text-to-speech
explained, 662
GetInstalledVoices method, 662

PromptBuilder class, 663-665
SelectVoice method, 662
SelectVoiceByHints method, 662
SetOutputToWaveFile method, 663
SpeakAsync method, 662
Speech Synthesis Markup Language (SSML), 663-665
SpeechSynthesizer, 662

TextBlock control
explained, 313-314
explicit versus implicit runs, 314
properties, 313
support for multiple lines of text, 315
whitespace, 314

TextBox control, 315

TextElement class, 319-320
Blocks
AnchoredBlock class, 326-327
BlockUIContainer, 321
List, 320
Paragraph, 320
sample code listing, 321-324
Section, 320
Table, 320
Inlines
AnchoredBlock, 326-327
defined, 324-325
InlineUIContainer, 329
LineBreak, 327
Span, 325-326

TextFormattingMode property (TextOptions), 312

TextHintingMode property (TextOptions), 312

TextOptions class, 312

TextRenderingMode property (TextOptions), 312

texture coordinates, 582

TextureCoordinates property (MeshGeometry3D class), 581

theme dictionaries, 466

theme styles, 70

ThemeDictionaryExtension, 468

ThemeInfoAttribute, 467-468

themes
defined, 415, 465
generic dictionaries, 467
per-theme styles and templates, 466-469
system colors, fonts, and parameters, 465-466
theme dictionaries, 466
theme styles, 70

Thickness class, 80-82

ThicknessConverter type converter, 82

thumb buttons (taskbar), adding, 232-233

ThumbButtonInfo property (TaskbarItemInfo), 232-233

thumbnail content (taskbar), customizing, 231

ThumbnailClipMargin property (TaskbarItemInfo), 231

tile brushes
DrawingBrush class, 518-522
ImageBrush class, 522-523
VisualBrush class, 523-525

TileMode enumeration, 521

TileMode property (DrawingBrush class), 519-521

Timelines, Storyboards as, 627-628

timer-based animation, 606-607

TimeSpan keyword, 51

To property (animation classes), 612-614

toast element, 778

toast notifications
audio, 778
canceling, 780
changing color of, 775
notification events, 778-779
prerequisites, 771-773
scheduled notifications, 779-780
sending, 774-775
templates, 775-778

ToastNotification object
Activated event, 778-779
Dismissed event, 779
Failed event, 779
sending, 774-775

ToastNotifier object, 774
ToggleButton class, 245-246
ToolBar control, 304-306
ToolBarOverflowPanel, 121
ToolBarPanel, 121
ToolBarTray, 121, 305
ToolTip class, 249-251
ToolTip property (ribbon controls), 295-296
ToolTipService class, 251
Top property (Canvas), 98
touch events, 159-162
basic touch events, 159-162
explained, 158
manipulation events
adding inertia with, 165-170
explained, 162-163
ManipulationCompleted, 163
ManipulationDelta, 163
ManipulationStarted, 163
ManipulationStarting, 163
panning/rotating/zooming with, 164-165
WPF 4 enhancements, 14
TouchDevice property (TouchEventArgs class), 159
TouchDown event, 160-162
TouchEventArgs class, 159
TouchMove event, 160-162
TouchUp event, 160-162
TraceSource object, 383
Transform method, 795
Transform property (Cameras), 547
Transform3Ds
combining, 560
explained, 552-553
house drawing example, 553-554
MatrixTransform3D class, 552, 560
RotateTransform3D class, 552, 557-560
ScaleTransform3D class, 552, 555-557
Transform3DGroup class, 552
TranslateTransform3D class, 552-555
TransformConverter type converter, 93
transforms
applying, 86-88
clipping and, 124
combining, 94
explained, 86
MatrixTransform, 93
RotateTransform, 88-90
ScaleTransform, 90-92
SkewTransform, 92
support for, 94-95
Transform3Ds
combining, 560
explained, 552-553
house drawing example, 553-554
MatrixTransform3D class, 552, 560
RotateTransform3D class, 552, 557-560
ScaleTransform3D class, 552, 555-557
Transform3DGroup class, 552
TranslateTransform3D class, 552-555
TranslateTransform, 92-93
TransformToAncestor method, 594-603
TransformToDescendant method, 598-603
transitions (animation), 645-649
Transitions property (VisualStateGroup class), 455
TranslateAccelerator method, 685-687
TranslateTransform, 92-93
TranslateTransform3D class, 554-555
Translation property (ManipulationDelta class), 163
transparent colors, 518
transparent regions, 153
trees
logical trees, 57-58
visual trees, 58-62
TreeView control, 302-304
TreeViewItem class, 303-304
TriangleIndices property (MeshGeometry3D class), 578-579
Trigger class. ***See*** **triggers**
TriggerBase class, 67

triggers
conflicting triggers, 429
in control templates, 432-434
data triggers, 67, 427-428
event triggers, 67
explained, 423-424
expressing logic with, 428
logical AND, 429-430
logical OR, 429
property triggers, 65-67, 424-427
respecting visual states with, 442-446
Triggers collection, 67
Triggers property (FrameworkElement class), 67
troubleshooting
data binding, 383
missing styles, 461
TryFindResource method, 357
tunneling, 143
turning off type conversion, 46
Twitter, pure-XAML Twitter client, 412-414
TwoWay binding, 403
type conversion, turning off, 46
type converters
BrushConverter, 28
explained, 26-27
finding, 28
FontSizeConverter, 28
GridLengthConverter, 114
ImageSourceConverter, 309
LengthConverter, 82
in procedural code, 27
ThicknessConverter, 82
TransformConverter, 93
values type-converted to object elements, 34
Type keyword, 52
TypeArguments keyword, 45, 51
typed styles, 421-422

U

UI Automation, supporting in custom controls, 745-746
UICulture element, 348
Uid keyword, 51, 349
UIElement class
binding to, 368
explained, 56
properties
IsKeyboardFocused, 152
IsMouseDirectlyOver, 153
RenderTransformOrigin property, 87
UIElement3D base class, 13
UIElement3D class
ContainerUIElement3D, 588
explained, 56
ModelUIElement3D, 586-588
uniform scale, 555
UniformGrid, 121
unmanaged code, mixing with managed code, 678
UpdateLayout method, 80
UpdateSourceExceptionFilter property (Binding object), 408
UpdateSourceTrigger enumeration, 404
UpdateSourceTrigger property (Binding object), 404
UpDirection property (Cameras), 546-548
Uri keyword, 51
URIs
package URI, 347
URIs for accessing binary resources, 344-345
usage context, 373
UseLayoutRounding property, 15
user controls
behavior, 721-723
dependency properties, 724-727
explained, 717-718
protecting controls from accidental usage, 723-724
routed events, 727-728
user controls versus custom controls, 718
user interfaces, 719-721

user interfaces
creating for PlayingCard custom control, 735-738
creating for user controls, 719-721
marking with localization IDs, 349
USER subsystems, 8

V

ValidatesOnNotifyDataErrors property (Binding object), 408-409
ValidateValueCallback delegate, 72
validation rules
adding to Binding object in, 405-409
INotifyDataErrorInfo interface, 408-409
ValidationRules property (Binding object), 406
value converters, 385
Binding.DoNothing values, 384
bridging incompatible data types, 380-383
customizing data display, 383-384
explained, 380
temporarily canceling data binding, 384
ValueMinMaxToIsLargeArcConverter, 445-446
ValueMinMaxToPointConverter, 445-446
Value value (NodeType property), 788
ValueMinMaxToIsLargeArcConverter, 445-446
ValueMinMaxToPointConverter, 445-446
ValueSource structure, 70
variables, HwndSource, 693-694
vector graphics, 10
verbosity of XAML, 53
versions of WPF
WPF 3.0, 12
WPF 3.5, 12-14
WPF 3.5 SP1, 13-14
WPF 4, 14-15
WPF 4.5, 12, 15-16
WPF Toolkit, 12
VerticalAlignment property
Canvas panel, 99
Grid, 120
StackPanel, 101
WrapPanel, 104
VerticalAlignment property (FrameworkElements class), 83-85
video support
controlling underlying media, 659-660
embedded resources, 661
explained, 656
MediaElement, 656-658
taking snapshots of individual video frames, 658
Windows Media Player, 656
VideoDrawing class, 474
Viewbox class, 127-130
Viewbox property (DrawingBrush class), 521-522
Viewport2DVisual3D class, 13, 588-589
Viewport3D class, 591-594
Viewport3DVisual class, 594
views
collection views, customizing
creating new views, 393-395
explained, 385
filtering, 391
grouping, 387-390
navigating, 391-392
sorting, 385-387
TreeView control, 302-304
viewSource_Filter method, 394
virtualization, 269, 276
VirtualizingPanel class, 102
VirtualizingStackPanel, 102, 259
Visibility property (FrameworkElements class), 82-83
Visible value (Visibility enumeration), 82
Visual C++, 678, 691
Visual class, 62
explained, 56
TransformToAncestor method, 594-598

visual effects, 527-529
visual hit testing
callback methods, 503
explained, 497
with multiple Visuals, 498-501
with overlapping Visuals, 501-503
simple hit testing, 497-498
Visual State Manager. *See* VSM
visual states
respecting with triggers, 442-446
respecting with VSM (Visual State Manager)
control parts, 447-449
control states, 449-455
Visual Studio debugger, Jump Lists and, 220
Visual Studio-like panes, creating
sequential states of user interface, 130-134
VisualStudioLikePanes.xaml, 134-136
VisualStudioLikePanes.xaml.cs, 136-140
Visual3D class
explained, 56
TransformToAncestor method, 598-603
TransformToDescendant method, 598-603
Visual3Ds
explained, 584
ModelVisual3D class, 585-586
UIElement3D class, 586
ContainerUIElement3D, 588
ModelUIElement3D, 586-588
Visual3D class
explained, 56
TransformToAncestor method, 598-603
TransformToDescendant method, 598-603
VisualBrush class, 523-525
VisualChildrenCount method, 495-496
Visuals
custom rendering, 497
displaying on screen, 494-496
DrawingContext methods, 492
DrawingVisuals
explained, 491
filling with content, 491-494
explained, 491
visual hit testing
callback methods, 503
explained, 497
with multiple Visuals, 498-501
with overlapping Visuals, 501-503
simple hit testing, 497-498
VisualStateGroup class, 455
VisualStateManager. *See* VSM
VisualStudioLikePanes.xaml file, 134-136
VisualStudioLikePanes.xaml.cs file, 136-140
VisualTransition objects, 645-649
VisualTreeHelper class, 59
vshost32.exe, 220
VSM (Visual State Manager), 14
animations and
button ControlTemplate with VisualStates, 641-644
transitions, 645-649
respecting visual states with
control parts, 447-449
control states, 449-455

W

WCF (Windows Communication Foundation), 13
Webcam control (Win32)
HostingWin32.cpp file, 681-683
Webcam.cpp file, 674-677
Webcam.h file, 674
Window1.h file, 679-680
Webcam.cpp file, 675-677
Webcam.h file, 674
websites, wpf.codeplex.com, 12
whitespace, TextBlock control, 314
Width property (FrameworkElements class), 78-80

Win32 controls, WPF interoperability
explained, 673-674
HwndSource class, 688-691
keyboard navigation, 683-687
launching modal dialogs, 688, 695
layout, 692-695
Webcam control, 674-683
winding order (mesh), 577-578
Window class, 178-180
Window1.h file, 679
Window1.xaml file, 713
Window1.xaml.cs file, 712
WindowHostingVisual.cs file, 493-495
WindowIconVisibility property (Ribbon), 281
WindowInteropHelper class, 704
Windows collection, 184
Windows Communication Foundation (WCF), 13
Windows desktop applications. *See also* Windows desktop features
multiple-document interface (MDI), 185
navigation-based desktop applications
explained, 193-194
hyperlinks, 197-198
journal, 198-200
Navigate method, 196-197
navigation containers, 194-196
navigation events, 200-201
Page elements, 194-196
returning data from pages, 203-204
sending data to pages, 202-203
single-instance applications, 186
standard desktop applications
Application class, 181-186
application state, 192
ClickOnce, 192-193
common dialogs, 188-189
custom dialogs, 189-190
explained, 177-178
multithreaded applications, 187
retrieving command-line arguments in, 184
splash screens, 187-188
Window class, 178-180
Windows Installer, 192
Windows Runtime APIs in, 771-772
Windows desktop features
Aero Glass, 233-236
Jump Lists
and Visual Studio debugger, 220
associating with applications, 218
explained, 218
JumpPaths, 225-228
JumpTasks, 218-219, 221-224
taskbar item customizations
explained, 230
taskbar item overlays, 231
taskbar item progress bars, 230
thumb buttons, 232-233
thumbnail content, 231
TaskDialogs, 236-239
Windows Forms, WPF interoperability
converting between two representatives, 703-704
ElementHost class, 700-702
explained, 695-696
launching modal dialogs, 699, 704
PropertyGrid, 696-699
Windows Installer, 192
Windows Media Player, 656
Windows Runtime APIs, 771-772
Windows themes, 470
Windows XP, WPF differences on, 16
WindowsFormsHost class, 698
WinFX, 24
WorkingDirectory property (JumpTask), 222
WPF 3.0, 12
WPF 3.5, 12-14
WPF 3.5 SP1, 13-14
WPF 4, 14-15
WPF 4.5, 12, 15-16
WPF Toolkit, 12

wpf.codeplex.com website, 12
WrapPanel
examples, 103-104
explained, 102
interaction with child layout properties, 104-105
properties, 102
right-to-left environments, 104
WriteableBitmap class, 13
writers (XAML)
explained, 783-785
node loops, 786-787
writing to live objects, 791-793
writing to XML, 793-794
XamlServices class, 794-797

X

X property
StylusPoint object, 157
TranslateTransform class, 92
x:Arguments keyword, 47, 49
x:Array keyword, 52
x:AsyncRecords keyword, 49
x:Boolean keyword, 49
x:Byte keyword, 49
x:Char keyword, 50
x:Class keyword, 40-41, 50
x:ClassAttributes keyword, 50
x:ClassModifier keyword, 50
x:Code keyword, 50
x:ConnectionId keyword, 50
x:Decimal keyword, 50
x:Double keyword, 50
x:FactoryMethod keyword, 47-50
x:FieldModifier keyword, 50
x:Int16 keyword, 50
x:Int32 keyword, 50
x:Int64 keyword, 50
x:Key keyword, 50, 352
x:Members keyword, 49-50, 52
x:Name keyword, 38, 50, 434
x:Null keyword, 52
x:Object keyword, 51
x:Property keyword, 49, 51
x:Reference keyword, 52, 699
x:Shared keyword, 51, 356
x:Single keyword, 51
x:Static keyword, 52
x:String keyword, 51
x:Subclass keyword, 51
x:SynchronousMode keyword, 51
x:TimeSpan keyword, 51
x:Type keyword, 52
x:TypeArguments keyword, 45, 51
x:Uid keyword, 51, 349
x:Uri keyword, 51
x:XData keyword, 51
XAML (Extensible Application Markup Language), 10
{} escape sequence, 375
accessing binary resources from, 343-346
advantages of, 18
advantages over procedural code, 20
animation
explained, 619-620
starting animations from property triggers, 626-627
Storyboards as Timelines, 627-628
TargetName property, 623-624
TargetProperty property, 620-623
BAML (Binary Application Markup Language)
decompiling back into XAML, 43-44
defined, 41
Binding object in, 363-365
CAML (Compiled Application Markup Language), 42
common complaints about, 52-53
compiling, 39-41
defined, 19-20

embedding PropertyGrid with, 698-699
explained, 17-18
extensibility, 35
factoring, 355-356
generated source code, 42
keywords, 49-52
loading and parsing at runtime, 36-38
loose XAML pages, 213-214
markup extensions
explained, 28-31
parameters, 29
in procedural code, 31
namespaces
explained, 22-24
implicit .NET namespaces, 23
mapping, 22
object elements
attributes, 21
content property, 31-32
declaring, 21
dictionaries, 33-34
explained, 20-21
lists, 32-33
naming, 38-39
processing child elements, 36
values type-converted to object elements, 34
order of property and event processing, 22
procedural code inside, 43
property elements, 25-26
pure-XAML Twitter client, 412-414
readers
explained, 783-785
markup compatibility, 791
node loops, 786-787
NodeType property, 787-788
sample XAML content, 789
XAML node stream, 789-791
XamlServices class, 794-797
running XAML examples, 18
specifications, 20
type converters
BrushConverter, 28
explained, 26-27
finding, 28
FontSizeConverter, 28
in procedural code, 27
values type-converted to object elements, 34
writers
explained, 783-785
node loops, 786-787
writing to live objects, 791-793
writing to XML, 793-794
XamlServices class, 794-797
XAML Browser Applications (XBAPs), 207
ClickOnce caching, 208
deployment, 211
explained, 207-208
full-trust XAML Browser applications, 210
integrated navigation, 210-211
limitations, 208-209
on-demand download, 212-213
security, 211
XAML2009
built-in data types, 46
dictionary keys, 46
event handler flexibility, 48
explained, 44-45
full generics support, 45
object instantiation via factory methods, 47-48
object instantiation with non-default constructors, 47
properties, defining, 49
XAML Browser Applications (XBAPs), 13, 207
ClickOnce caching, 208
deployment, 211
explained, 207-208
full-trust XAML Browser applications, 210

integrated navigation, 210-211
limitations, 208-209
on-demand download, 212-213
security, 211
XAML Cruncher, 19
XAML2009
built-in data types, 46
dictionary keys, 46
event handler flexibility, 48
explained, 44-45
full generics support, 45
object instantiation via factory methods, 47-48
object instantiation with non-default constructors, 47
properties, defining, 49
XamlBackgroundReader class, 784
XamlMember class, 788
XamlObjectReader class, 783
XamlObjectWriter class, 784
XamlPad, 19
XAMLPAD2009, 18-19
XamlPadX, 19, 59
XamlReader class
explained, 783-785
Load method, 36-37
LoadAsync method, 37
XamlServices class, 794-797
XamlType class, 788
XamlWriter class, 44, 783-785
XamlXmlReader class, 783-786
markup compatibility, 791
sample XAML content, 789
XAML node stream, 789-791
XamlXmlWriter class, 784
XBAPs. *See* XAML Browser Applications
XData keyword, 51
XML, writing to, 793-794
XML Paper Specification (XPS), 319
xml:lang attibute, 49
xml:space attribute, 49
XmlDataProvider class, 396-400
XPath (XML PAth Language), 397
XPS (XML Paper Specification), 319

Y-Z

Y property
StylusPoint object, 157
TranslateTransform class, 92

Z order, 99-100
Z-fighting, 543
zooming
enabling with touch events, 164-165
with inertia, 166-167