The #1 WPF Book—Now Updated for WPF 4!

Adam Nathan

Full Color
Code samples appear as they do in Visual Studio!

WPF 4

UNLEASHED

SAMS

WPF 4 Unleashed

ISBN-13: 978-0-672-33119-0
ISBN-10: 0-672-33119-5

Library of Congress Cataloging-in-Publication Data

Nathan, Adam.
WPF 4 unleashed / Adam Nathan.
p. cm.
Includes index.
ISBN 978-0-672-33119-0
1. Windows presentation foundation. 2. Application software. 3. Microsoft .NET Framework. I. Title.
QA76.76.A65N386 2010
006.7'882—dc22

2010017765

Printed in the United States on America

First Printing June 2010

Trademarks

Warning and Disclaimer

Bulk Sales

Editor-in-Chief
Karen Gettman

Executive Editor
Neil Rowe

Development Editor
Mark Renfrow

Managing Editor
Kristy Hart

Project Editor
Betsy Harris

Copy Editor
Kitty Wilson

Indexer
Erika Millen

Proofreader
Kathy Ruiz

Technical Editors
Dwayne Need
Robert Hogue
Joe Castro
Jordan Parker

Publishing Coordinator
Cindy Teeters

Book Designer
Gary Adair

Composition
Bronkella Publishing LLC

Introduction

Thank you for picking up *WPF 4 Unleashed*! Windows Presentation Foundation (WPF) is Microsoft's premier technology for creating Windows graphical user interfaces, 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 Silverlight, which has extended WPF technology onto the Web and into devices such as Windows phones.

Ever since WPF was publicly announced in 2003 (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, released in April 2010, improves on previous versions of WPF in just about every dimension.

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 edition of this book was far more successful than I ever imagined it would be. Now, almost four years later, I believe that this second edition accomplishes all the same

goals but with even more depth. In addition to covering new features introduced in WPF 3.5, WPF 3.5 SP1, and WPF 4, it expands the coverage of the existing features from the first version of WPF. 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 Windows. 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, reading this book can also make you a better developer for Windows Phone 7 and even a better web developer.

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 Microsoft Expression Blend.

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
- Highlights the latest features, such as multi-touch, text rendering improvements, new controls, XAML language enhancements, the Visual State Manager, easing functions, and much 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, such as galleries, ScreenTips, custom control layouts, and more
- 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 new Windows 7 features in WPF applications, such as Jump Lists, and how to go beyond some of the limitations of WPF

This book doesn't cover every last bit of WPF. (In particular, XML Paper Specification [XPS] documents 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 the final release of version 4.0 of Windows Presentation Foundation, the corresponding Windows SDK, and Visual Studio 2010.

The following software is required:

- A version of Windows that supports the .NET Framework 4.0. This can be Windows XP with Service Pack 2 (including Media Center, Tablet PC, and x64 editions), Windows Server 2003 with Service Pack 1 (including the R2 edition), Windows Vista, or later versions.
- The .NET Framework 4.0, which is installed by default starting with Windows Vista. For earlier versions of Windows, you can download the .NET Framework 4.0 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 2010 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, Microsoft Expression (specifically Expression Blend) can be extremely helpful.

A few examples are specific to Windows Vista, Windows 7, or a computer that supports multi-touch, but 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/9780672331190 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, and What About Silverlight?
- 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 Multi-Touch
- Chapter 7: Structuring and Deploying an Application
- Chapter 8: Exploiting Windows 7

Part II equips you with the knowledge to assemble and deploy a traditional-looking application (although some fancier effects, such as transforms, nonrectangular windows, and Aero Glass, are also covered). Chapters 4 and 5 discuss arranging controls (and other elements) in a user interface. Chapter 6 covers input events, including new support for engaging multi-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 Windows 7 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

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.

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

Digging Deeper Sidebars

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.

CHAPTER 1

Why WPF, and What About Silverlight?

IN THIS CHAPTER

In movies and on TV, the main characters are typically an exaggeration of the people you encounter in real life. They're more attractive, they react more quickly, and they somehow always know exactly what to do. The same could be said about the software they use.

This first struck me back in 1994 when watching the movie *Disclosure*, starring Michael Douglas, Demi Moore, and an email program that looks nothing like Microsoft Outlook! Throughout the movie, we're treated to various visual features of the program: a spinning three-dimensional "e," messages that unfold when you open them and crumple when you delete them, hints of inking support, and slick animations when you print messages. (The email program isn't even the most unrealistic software in the movie. I'll just say "virtual reality database" and leave it at that.)

Usability issues aside, Hollywood has been telling us for a long time that software in the real world isn't as compelling as it should be. You can probably think of several examples on your own of TV shows and movies with comically unrealistic software. But lately, real-world software has been catching up to Hollywood's standards! You can already see it in traditional operating systems (yes, even in Windows), on the web, and in software for devices such as the iPhone, iPad, Zune, TiVo, Wii, Xbox, Windows phones, and many more. Users have increasing expectations for the experience of using software, and companies are spending a great deal of time and money on user interfaces that differentiate themselves from the competition. This isn't limited to consumer-facing software; even business applications and internal tools can greatly benefit from a polished user interface.

With higher demands placed on user interfaces, traditional software development processes and technologies often fall short. Modern software usually needs to support rapid iteration and major user interface changes throughout the process—whether such changes are driven by professional graphic designers, developers with a knack for designing user interfaces, or a boss who wants the product to be more "shiny" and animated. For this to be successful, you need technology and tools that make it natural to separate the user interface from the rest of the implementation as much as possible and to decouple visual behavior from the underlying program logic. Developers should be able to create a fully functional "ugly" application that designers can directly retheme without requiring developers to translate their artwork. The Win32 style of programming, in which controls directly contain code to paint and repaint themselves, makes rapid user interface iteration far too difficult for most projects.

In 2006, Microsoft released a technology to help people create 21st-century software that meets these high demands: Windows Presentation Foundation (WPF). With the release of WPF 4 in 2010, the technology is better than ever at delivering amazing results for just about any kind of software. Almost a decade after Tom Cruise helped popularize the idea of multi-touch computer input in the movie *Minority Report*, and after successful multi-touch implementations in a variety of devices (most notably the iPhone), WPF 4 and Windows 7 are bringing multi-touch to the masses. Hollywood better start coming up with some fresh ideas!

A Look at the Past

The primary technologies behind many Windows-based user interfaces—the graphics device interface (GDI) and USER subsystems—were introduced with Windows 1.0 in 1985. That's almost prehistoric in the world of technology! In the early 1990s, OpenGL (created by Silicon Graphics) became a popular graphics library for doing advanced two-dimensional (2D) and three-dimensional (3D) graphics on both Windows and non-Windows systems. This was leveraged by people creating computer-aided design (CAD) programs, scientific visualization programs, and games. DirectX, a Microsoft technology introduced in 1995, provided a new high-performance alternative for 2D graphics, input, communication, sound, and eventually 3D (introduced with DirectX 2 in 1996).

Over the years, many enhancements have been made to both GDI and DirectX. GDI+, introduced in the Windows XP time frame, tried to improve upon GDI by adding support for features such as alpha blending and gradient brushes. It ended up being slower than GDI due to its complexity and lack of hardware acceleration. DirectX (which, by the way, is the technology behind Xbox) continually comes out with new versions that push the limits of what can be done with computer graphics. With the introduction of .NET and managed code in 2002, developers were treated to a highly productive model for creating Windows (and web) applications. In this world, Windows Forms (built on top of GDI+) became the primary way a C#, Visual Basic, and (to a lesser degree) C++ developer started to create new user interfaces on Windows. Windows Forms has been a successful and productive technology, but it still has all the fundamental limitations of GDI+ and USER.

Starting with DirectX 9, Microsoft shipped a DirectX framework for managed code (much like it shipped libraries specifically for Visual Basic in the past), which eventually was supplanted by the XNA Framework. Although this enables C# developers to use DirectX without most of the complications of .NET/COM interoperability, these managed frameworks aren't significantly easier to use than their unmanaged counterparts unless you're writing a game. (The XNA Framework makes writing a game easier because it includes new libraries specifically for game development and works with compelling tools such as the XNA Framework Content Pipeline and XNA Game Studio Express.)

So although you could have developed a Windows-based email program with the 3D effects seen in *Disclosure* ever since the mid-1990s with non-GDI technologies (actually, probably mixing DirectX or OpenGL with GDI), such technologies are rarely used in mainstream Windows applications even more than a decade later. There are several reasons for this: The hardware required to get a decent experience hasn't been ubiquitous until recently, it has been at least an order of magnitude harder to use alternative technologies, and GDI-based experiences have been considered "good enough."

Graphics hardware continues to get better and cheaper and consumer expectations continue to rise, but until WPF, the difficulty of creating modern user experiences had not been addressed. Some developers would take matters into their own hands to get cooler-looking applications and controls on Windows. A simple example of this is using bitmaps for buttons instead of using the standard button control. These types of customizations can not only be expensive to develop, but they also often produce a flakier experience. Such applications often aren't as accessible as they should be, don't handle high dots-per-inch (DPI) settings very well, and have other visual glitches.

Enter WPF

Microsoft recognized that something brand new was needed that escaped the limitations of GDI+ and USER yet provided the kind of productivity that people enjoy with frameworks like Windows Forms. And with the continual rise of cross-platform applications based on HTML and JavaScript, Windows desperately needed a technology that's as fun and easy to use as these, yet with the power to exploit the capabilities of the local computer. Windows Presentation Foundation (WPF) is the answer for software developers and graphic designers who want to create modern user experiences without having to master several difficult technologies. Although "Presentation" sounds like a lofty term for what I would simply call a user interface, it's probably more appropriate for describing the higher level of visual polish that's expected of today's applications and the wide range of functionality included in WPF!

The highlights of WPF include the following:

- **Broad integration**—Prior to WPF, a Windows developer who wanted to use 3D, video, speech, and rich document viewing in addition to normal 2D graphics and controls would have to learn several independent technologies with a number of inconsistencies and attempt to blend them together without much built-in support. But WPF covers all these areas with a consistent programming model as well as tight integration when each type of media gets composited and rendered. You can apply

the same kind of effects consistently across different media types, and many of the techniques you learn in one area apply to all the other areas.

- **Resolution independence**—Imagine a world in which moving to a higher resolution or DPI setting doesn't mean that everything gets smaller; instead, graphics and text simply get crisper! Envision user interfaces that look reasonable on a small netbook as well as on a 60-inch TV! WPF makes this easy and gives you the power to shrink or enlarge elements on the screen independently from the screen's resolution. A lot of this is possible because of WPF's emphasis on vector graphics.
- **Hardware acceleration**—WPF is built on Direct3D, so content in a WPF application—whether 2D or 3D, graphics, or text—is converted to 3D triangles, textures, and other Direct3D objects and then rendered by hardware. This means that WPF applications get the benefits of hardware acceleration for smoother graphics and all-around better performance (due to work being offloaded to graphics processing units [GPUs] instead of central processor units [CPUs]). It also ensures that all WPF applications (not just high-end games) receive benefits from new hardware and drivers, whose advances typically focus on 3D capabilities. But WPF doesn't *require* high-end graphics hardware; it has a software rendering pipeline as well. This enables features not yet supported by hardware, enables high-fidelity printing of any content on the screen, and is used as a fallback mechanism when encountering inadequate hardware resources (such as an outdated graphics card or even a high-end one that has simply run out of GPU resources such as video memory).
- **Declarative programming**—Declarative programming is not unique to WPF, as Win16/Win32 programs have used declarative resource scripts to define the layout of dialog boxes and menus for over 25 years. And .NET programs of all types often leverage declarative custom attributes plus configuration and resource files based on Extensible Markup Language (XML). But WPF takes declarative programming to the next level with Extensible *Application* Markup Language (XAML; pronounced "Zammel"). The combination of WPF and XAML is similar to using HTML to define a user interface—but with an incredible range of expressiveness. This expressiveness even extends beyond the bounds of user interfaces; WPF uses XAML as a document format, a representation of 3D models, and more. The result is that graphic designers are empowered to contribute directly to the look and feel of applications, as well as some behavior for which you'd typically expect to have to write code. The next chapter examines XAML in depth.
- **Rich composition and customization**—WPF controls can be composed in ways never before seen. You can create a `ComboBox` filled with animated `Buttons` or a `Menu` filled with live video clips! Although these particular customizations might sound horrible, it's important that you don't have to write a bunch of code (or any code!) to customize controls in ways that the control authors never imagined (unlike owner-draw in prior technologies). Along the same lines, WPF makes it quite easy to "skin" applications with radically different looks (covered in Chapter 14, "Styles, Templates, Skins, and Themes").

In short, WPF aims to combine the best attributes of systems such as DirectX (3D and hardware acceleration), Windows Forms (developer productivity), Adobe Flash (powerful animation support), and HTML (declarative markup). With the help of this book, I think you'll find that WPF gives you more productivity, power, and fun than any other technology you've worked with in the past!

DIGGING DEEPER

GDI and Hardware Acceleration

GDI is actually hardware accelerated on Windows XP. The video driver model explicitly supported accelerating common GDI operations. Windows Vista introduced a new video driver model that does not hardware accelerate GDI primitives. Instead, it uses a "canonical display device" software implementation of the legacy video driver for GDI. However, Windows 7 reintroduced partial hardware acceleration for GDI primitives.

FAQ

Does WPF enable me to do something that I couldn't have previously done?

Technically, the answer is "No," just like C# and the .NET Framework don't enable you to do something that you couldn't do in assembly code. It's just a question of how much work you want to do to get the desired results!

If you were to attempt to build a WPF-equivalent application from scratch without WPF, you'd not only have to worry about the drawing of pixels on the screen and interaction with input devices, you'd also need to do a ton of additional work to get the accessibility and localization support that's built in to WPF, and so on. WPF also provides the easiest way to take advantage of Windows 7 features, such as defining Jump List items with a small chunk of XAML (see Chapter 8, "Exploiting Windows 7").

So I think most people would agree that the answer is "Yes" when you factor time and money into the equation!

FAQ

When should I use DirectX instead of WPF?

DirectX is more appropriate than WPF for advanced developers writing hard-core "twitch games" or applications with complex 3D models where you need maximum performance. That said, it's easy to write a naive DirectX application that performs far worse than a similar WPF application.

DirectX is a low-level interface to the graphics hardware that exposes all the quirks of whatever GPU a particular computer has. DirectX can be thought of as assembly language in the world of graphics: You can do anything the GPU supports, but it's up to you (the application author) to support all the hardware variations. This is onerous, but such low-level hardware access enables skilled developers to make their own tradeoffs between fine-grained quality and speed. In addition, DirectX exposes cutting-edge features of GPUs as they emerge more quickly than they appear in WPF.

Continued

In contrast, WPF provides a high-level abstraction that takes a description of a scene and figures out the best way to render it, given the hardware resources available. (It's a *retained mode* system rather than an *immediate mode* system.) 2D is the primary focus of WPF; its 3D support is focused on data visualization scenarios and integration with 2D rather than supporting the full power of DirectX.

The downside of choosing DirectX over WPF is a potentially astronomical increase in development cost. A large part of this cost is the requirement to test an application on each driver/GPU combination you intend to support. One of the major benefits of building on top of the WPF is that Microsoft has already done this testing for you! You can instead focus your testing on low-end hardware for measuring performance. The fact that WPF applications can even leverage the client GPU in a partial-trust environment is also a compelling differentiator.

Note that you are able to use both DirectX and WPF in the same application. Chapter 19, "Interoperability with Non-WPF Technologies," shows how this can be done.

The Evolution of WPF

Oddly enough, WPF 4 is the fourth major release of WPF. It's odd because the first release had the version number 3.0! The first release in November 2006 was called WPF 3.0 because it shipped as part of the .NET Framework 3.0. The second release—WPF 3.5—came almost exactly a year later (one day shy, in fact). The third release, once again, came almost a year later (in August 2008). This release was a part of Service Pack 1 (SP1) for .NET 3.5, but this was no ordinary service pack as far as WPF was concerned—it contained many new features and improvements.

In addition to these major releases, Microsoft introduced a "WPF Toolkit" in August 2008 at http://wpf.codeplex.com that, along with miscellaneous tools and samples, gets updated several times a year. The WPF Toolkit has been used as a way to ship features more quickly and in an experimental form (often with full source code). Features introduced in the WPF Toolkit often "graduate" to get included in a future release of WPF, based on customer feedback about their desirability and readiness.

When the first version of WPF was released, tool support was almost nonexistent. The following months brought primitive WPF extensions for Visual Studio 2005 and the first public preview release of Expression Blend. Now, Visual Studio 2010 not only has first-class support for WPF development but has been substantially rewritten to be a WPF application itself! Expression Blend, an application built 100% with WPF, has also gained a lot of functionality for designing and prototyping great user interfaces. And in the past several years, numerous WPF-based applications have been released from companies such as Autodesk, SAP, Disney, Blockbuster, Roxio, AMD, Hewlett Packard, Lenovo, and many more. Microsoft itself, of course, has a long list of software built with WPF (Visual Studio, Expression, Test and Lab Manager, Deep Zoom Composer, Songsmith, Surface, Semblio, Robotics Studio, LifeCam, Amalga, Games for Windows LIVE Marketplace, Office

TIP

To inspect the WPF elements used in any WPF-based application, you can use the Snoop tool available from http://snoopwpf.codeplex.com.

Communicator Attendant, Active Directory Administrative Center, Dynamics NAV, Pivot, PowerShell ISE, and many more).

Let's take a closer look at how WPF has changed over time.

Enhancements in WPF 3.5 and WPF 3.5 SP1

The following notable changes were made to WPF in versions 3.5 and 3.5 SP1:

- **Interactive 3D**—The worlds of 2D and 3D were woven together even more seamlessly with the `UIElement3D` base class, which gives 3D elements input, focus, and events; the odd-sounding `Viewport2DVisual3D` class, which can place any interactive 2D controls inside a 3D scene; and more. See Chapter 16, "3D Graphics."
- **First-class interoperability with DirectX**—Previously, WPF applications could only interoperate with DirectX via the lowest common denominator of Win32. Now, WPF has functionality for interacting directly with Direct3D surfaces with the `D3DImage` class rather than being forced to interact with its host HWND. One benefit from this is the ability to place WPF content on top of DirectX content and vice versa. See Chapter 19.
- **Better data binding**—WPF gained support for XLINQ binding, better validation and debugging, and output string formatting in XAML that reduces the need for custom procedural code. See Chapter 13, "Data Binding."
- **Better special effects**—The first version of WPF shipped with a handful of bitmap effects (blur, drop shadow, outer glow, emboss, and bevel) but with a warning to not use them because their performance was so poor! This has changed, with a new set of hardware-accelerated effects and a whole new architecture that allows you to plug in your own custom hardware-accelerated effects via pixel shaders. See Chapter 15, "2D Graphics."
- **High-performance custom drawing**—WPF didn't previously have a good answer for custom drawings that involve thousands of points or shapes, as even the lowest-level drawing primitives have too much overhead to make such things perform well. The `WriteableBitmap` class was enhanced so you can now specify dirty regions when drawing on it rather than getting a whole new bitmap every frame! Because `WriteableBitmap` only lets you set pixels, it is a very primitive form of "drawing," however.
- **Text improvements**—There's now better performance, better international support (improved input method editor [IME] support and improved Indic script support), and enhancements to `TextBox` and `RichTextBox`. See Chapter 11, "Images, Text, and Other Controls."
- **Enhancements to partial-trust applications**—More functionality became available in the partial-trust sandbox for .NET applications, such as the ability to use Windows Communication Foundation (WCF) for web service calls (via `basicHttpBinding`) and the ability to read and write HTTP cookies. Also, support for XAML Browser Applications (XBAPs)—the primary mechanism for running partial-trust

WPF applications—was extended to the Firefox web browser instead of just Internet Explorer (In WPF, however, the add-on that enables this is no longer installed by default.)

- **Improved deployment for applications and the .NET Framework**—This arrived in many forms: a smaller and faster .NET Framework installation process thanks to the beginnings of a .NET Framework "client profile" that excludes server-only .NET pieces such as ASP.NET; a new "bootstrapper" component that handles all .NET Framework dependencies, installations, and upgrades for you as well as enabling setups with custom branding; and a variety of new ClickOnce features.
- **Improved performance**—WPF and the underlying common language runtime implemented several changes that significantly boosted the performance of WPF applications without any code changes needed. For example, the load time (especially first-time load) has been dramatically improved, animations (especially slow ones) are much smoother, data binding is faster in a number of scenarios, and layered windows (described in Chapter 8) are now hardware accelerated. Other performance improvements were made that you must opt into due to compatibility constraints, such as improved virtualization and deferred scrolling in items controls, described in Chapter 10, "Items Controls."

Enhancements in WPF 4

WPF 4 brings the following changes, on top of the changes from previous versions:

- **Multi-touch support**—When running on computers that support multi-touch and run Windows 7 or later, WPF elements can get a variety of input events, from low-level data, to easy-to-consume manipulations (such as rotation and scaling), to high-level—including custom—gestures. The built-in WPF controls have also been updated to be multi-touch aware. The WPF team leveraged the work previously done by the Microsoft Surface team (whose software is built on WPF). As a result, multi-touch in WPF 4 is compatible with version 2 of the Surface SDK, which is great news for anyone considering developing for both Windows and Surface. See Chapter 6, "Input Events: Keyboard, Mouse, Stylus, and Multi-Touch."
- **First-class support for other Windows 7 features**—Multi-touch is a cool new feature of Windows 7, but there are plenty of others that don't require special hardware—so many more users will appreciate their inclusion. WPF provides the best way to integrate with new taskbar features such as Jump Lists and icon overlays, integrate with the latest common dialogs, and more. See Chapter 8.
- **New controls**—WPF 4 includes controls such as `DataGrid`, `Calendar`, and `DatePicker`, which originally debuted in the WPF Toolkit. See Chapter 11.
- **Easing animation functions**—Eleven new animation classes such as `BounceEase`, `ElasticEase`, and `SineEase` enable sophisticated animations with custom rates of acceleration and deceleration to be performed completely declaratively. These "easing functions" and their infrastructure were first introduced in Silverlight 3 before being adopted by WPF 4.

- **Enhanced styling with Visual State Manager**—The Visual State Manager, originally introduced in Silverlight 2, provides a new way to organize visuals and their interactivity into "visual states" and "state transitions." This feature makes it easier for designers to work with controls in tools such as Expression Blend, but importantly also makes it easier to share templates between WPF and Silverlight.
- **Improved layout on pixel boundaries**—WPF straddles the line between being automatically DPI independent (which requires ignoring physical pixel boundaries) and having visual elements that look crisp (which, especially for small elements, requires being aligned on pixel boundaries). From the beginning, WPF has supported a property called `SnapsToDevicePixels` that forces "pixel snapping" on elements. But using `SnapsToDevicePixels` can be complex and doesn't help in some scenarios. Silverlight went back to the drawing board and created a property called `UseLayoutRounding` that works more naturally. WPF 4 now has this property. Just set it to `true` on a root element, and the positions of that element plus all of children will be rounded up or down to lie on pixel boundaries. The result is user interfaces that can scale *and* can easily be crisp!
- **Non-blurry text**—WPF's emphasis on DPI independence and a scalable user interface has been an issue for small text—the kind of text that occurs a lot in traditional user interfaces on 96-DPI screens. This has frustrated numerous users and developers. In fact, I've always claimed that I can spot a user interface created with WPF simply by looking at the blurriness of its text. WPF 4 has finally addressed this with an alternative way to render text that can make it look as crisp as GDI-based text yet with almost all the benefits that WPF brings. Visual Studio 2010, for example, uses this rendering mode for its text documents. Because there are some limitations to the new rendering approach, you must opt into it. See Chapter 11.
- **More deployment improvements**—The .NET Framework client profile can run side-by-side with the full .NET Framework, and it can be used in just about every scenario relevant for WPF applications. In fact, .NET 4.0 projects in Visual Studio 2010 target the smaller client profile by default.
- **More performance improvements**—In order to make vector graphics perform as well as possible, WPF can cache rendered results as bitmaps and reuse them. For advanced scenarios, you can control this behavior with the new `CacheMode` property. See Chapter 15. The heavy usage of WPF in Visual Studio 2010 drove a lot of miscellaneous performance improvements into WPF 4 across the board, but all WPF applications get to enjoy these improvements.

FAQ

What will be added to WPF after version 4?

Nothing has been announced at the time of writing, but I think it's safe to say that performance and increased synergy with Silverlight will continue to be two major themes of WPF's evolution. Plus, the WPF Toolkit provides some clues to future features that could be integrated into the core platform, such as chart controls, a `BreadcrumbBar` control, a `NumericUpDown` control, and more.

FAQ

Are there any differences with WPF, depending on the version of Windows?

WPF exposes APIs that are relevant only for Windows 7 and later, such as multi-touch functionality and various features described in Chapter 8. Besides that, WPF has a few behavioral differences when running on Windows XP (the oldest version of Windows that WPF supports). For example, 3D objects do not get antialiased.

And, of course, WPF controls have different default themes to match their host operating system (Aero on Windows Vista and Windows 7 versus Luna on Windows XP).

Windows XP also has an older driver model that can negatively impact WPF applications. The driver model in later versions of Windows virtualizes and schedules GPU resources, making a system perform better when multiple GPU-intensive programs are running. Running multiple WPF or DirectX applications might bog down a Windows XP system but shouldn't cause performance issues on more recent versions of Windows.

What About Silverlight?

Silverlight is a small, lightweight version of the .NET Framework targeted at rich web scenarios (as an alternative to Adobe Flash and Flex, for example). Silverlight chose to follow WPF's approach to creating user interfaces rather than creating yet another distinct technology—and this approach has some great benefits. It was first released in 2007 and, like WPF, is already in its fourth major version. Silverlight 4 was released in April 2010, a few days after the release of WPF 4.

The relationship between WPF and Silverlight is a bit complex, and there is some confusion about when to use one technology versus the other. This is further exacerbated by the fact that WPF applications can run inside a web browser (as XBAPs) and be just as "web based" as Silverlight applications, and Silverlight applications can run outside a web browser, even in an offline mode.

Silverlight is mostly a subset of WPF plus the most fundamental classes in the .NET Framework (core data types, collection classes, and so on). Each new version of Silverlight includes more and more WPF functionality. Although compatibility with WPF and the full .NET Framework is a goal for Silverlight, its creators have taken some opportunities to learn from mistakes made in WPF and the .NET Framework. They have made some changes and begun to support new features that don't yet exist in the full .NET Framework. Some of these changes or additions have been later adopted by WPF and the full .NET Framework (such as the Visual State Manager and layout rounding), but others have not (such as video brushes and perspective transforms). There are parts of WPF and the .NET Framework that Silverlight will probably never support.

The bottom line is that the question to ask yourself isn't "Should I use WPF or Silverlight?" but rather, "Should I use the full .NET Framework or the small .NET Framework?" If you will require functionality that exists only in the full .NET Framework, then the choice is pretty simple. And WPF is the recommended user interface technology to use with the full .NET Framework. Similarly, if the ability to run on a Mac or devices

other than a standard PC is a requirement, then the choice is also clear. And Silverlight has only one user interface technology (although it interoperates with HTML nicely). Otherwise, the best choice depends greatly on the nature of the software and the target audience.

Ideally, you wouldn't have to make an up-front choice of which framework you want to target. Ideally, you could use the same codebase—even the same compiled binaries—and have an easy way to morph the application to exploit capabilities of the underlying device, whether your program is running on a mobile device, a full Windows PC, or a Mac. Maybe one day that will be true, but in the meantime, having a common codebase that can work for both WPF and Silverlight involves a bit of work. The most common approach has been to create a Silverlight-compatible codebase with `#ifdef` blocks for WPF-specific functionality, so you can compile separately for Silverlight versus WPF with minimal divergence in code.

It is my expectation (and hope) that the distinction between WPF and Silverlight will fade over time. While Silverlight is a much cooler name than Windows Presentation Foundation, the fact that these technologies have different names causes trouble and artificial distinctions. The way to think of Silverlight and WPF is as two implementations of the same basic technology. In fact, inside Microsoft, largely the same team works on both. Microsoft talks a lot about having a "client continuum" to target all platforms and devices with common skills (what you learn in this book), common tools (Visual Studio, Expression Blend, and others), and at least common *code* (a .NET language such as C# or VB along with XAML, for example) if not common *binaries*. While it would be overkill to call this book *WPF and Silverlight Unleashed*, it should be comforting to know that the knowledge you gain from this book can help you be an expert in both WPF and Silverlight.

Summary

As time passes, more software is delivering high-quality—sometimes *cinematic*—experiences, and software that doesn't risks looking old-fashioned. However, the effort involved in creating such user interfaces—especially ones that exploit Windows—has been far too difficult in the past.

WPF makes it easier than ever before to create all kinds of user interfaces, whether you want to create a traditional-looking Windows application or an immersive 3D experience worthy of a role in a summer blockbuster. Such a rich user interface can be evolved fairly independently from the rest of an application, allowing graphic designers to participate in the software development process much more effectively. But don't just take my word for it; read on to see for yourself how it's done!

Symbols/Numbers

A

B

C

D

E

F

G

H

I

J

K

L

M

N

P

Q

R

S

T

U

V

W

X

Y-Z