Preface

OVER THE PAST nine years I have worked on many user interface (UI) projects at Microsoft. I have spent time working on Visual Basic 6.0, the version of Windows Foundation Classes that shipped with Visual J++ 6.0, Windows Forms for the .NET Framework, internal projects that never saw the light of day, and now, finally, Windows Presentation Foundation (WPF).

I started working on WPF about 18 months after the team was created, joining as an architect in the fall of 2002. At that time, and until late 2005, the team and technology were code-named *Avalon.* Early in 2003 I had the privilege of helping to redesign the platform, which we released as a technology preview for the Professional Developers Conference (PDC) 2003 in Los Angeles. WPF is the product of almost five years of work by more than 300 people. Some of the design ideas in WPF date back to products from as early as 1997 (Application Foundation Classes for Java was the beginning of some of the ideas for creating components in WPF).

When I joined the WPF team, it was still very much in research mode. The project contained many more ideas than could possibly ship in a single version. The primary goal of WPF—to replace all the existing infrastructure for building applications on the client with a new integrated platform that would combine the best of Win32 and the Web—was amazingly ambitious and blurred the lines between user interface, documents, and media. Over the years we have made painful cuts, added great features, and listened to a ton of feedback from customers, but we never lost sight of that vision.

A Brief History of GUI

Graphical user interfaces (GUIs) started in the early 1980s in the Xerox PARC laboratory. Since then, Microsoft, Apple, and many other companies have created many platforms for producing GUI applications. Microsoft's GUI platform began with Windows 1.0 but didn't gain widespread use until Windows 3.0 was released in 1990. The primary programming model for building GUI applications consisted of the two dynamic link libraries (DLLs): User and GDI. In 1991 Microsoft released Visual Basic 1.0, which was built on top of User and GDI, and offered a much simpler programming model.

Visual Basic's UI model, internally called *Ruby,*[1] was far simpler to use than were the raw Windows APIs. This simplicity angered the developers who felt that programming should be difficult. The early versions of Visual Basic were significantly limited, however, so most developers building "real" applications chose to program directly to User and GDI. Over time, that changed. By the time the Microsoft world moved to 32-bit with the release of Windows 95 and Visual Basic 4.0, the VB crowd was gaining significant momentum and was offering a much wider breadth of platform features.

At about the same time there was another big shift in the market: the Internet. Microsoft had been working on a replacement for the Visual Basic UI model that was internally called *Forms3.* For various reasons, Microsoft decided to use this model as the basis for an offering in the browser space. The engine was renamed *Trident* internally, and today it ships in Windows as `MSHTML.dll`. Trident evolved over the years to be an HTML-specific engine with great text layout, markup, and scripting support.

Also around the same time, another phenomenon appeared on everyone's radar: managed code. Visual Basic had been running in a managed environment for a long time (as had many other languages), but the introduction of Java by Sun Microsystems in 1994 marked the first time that many developers were exposed to the notion of a virtual machine. Over the next several years managed code became a larger and larger force in the market,

1. This code name has no relationship to the Ruby programming language.

and in 2002 Microsoft released its own general-purpose managed-code platform: the .NET Framework. Included in the .NET Framework was Windows Forms, a managed-code API for programming User32 and GDI+ (a successor to GDI32). Windows Forms was intended to replace the old Ruby forms package in Visual Basic.

As we entered the new millennium, Microsoft had four predominant UI platforms: User32/GDI32, Ruby, Trident, and Windows Forms. These technologies solve different sets of problems, have different programming models, and are used by different sets of customers. Graphics systems had also evolved: In 1995, Microsoft introduced DirectX, a graphics system that gave the programmer much deeper access to the hardware. But none of the four main UI technologies used this newfound power in a meaningful way.

There was a real problem to be solved here. Customers were demanding the richness of modern video games and television productions in their applications. Media, animation, and rich graphics should be everywhere. They wanted rich text support because almost every application displayed some type of text or documentation. They wanted rich widgets for creating applications, buttons, trees, lists, and text editors—all of which were needed to build the most basic application.

With these four major platforms a large percentage of the customers' needs were met, but they were all islands. The ability to mix and match parts of the platforms was difficult and error-prone. From a purely selfish point of view, Microsoft management (well, I'll name names: Bill Gates) was tired of paying four teams to build largely overlapping technologies.

In 2001, Microsoft formed a new team with a simple-sounding mission: to build a unified presentation platform that could eventually replace User32/GDI32, Ruby, Trident, and Windows Forms, while enabling the new scenarios that customers were demanding in the presentation space. The people who made up this team came largely from the existing presentation platform teams, and the goal was to produce a best-of-breed platform that could really be a quantum leap forward.

And so the Avalon team was formed. At PDC 2003, Microsoft announced Avalon (the code name at the time). Later the project was given the name Windows Presentation Foundation.

Principles of WPF

WPF has taken a long time to build, but for the entire life of this project, several guiding principles have remained constant.

Build a Platform for Rich Presentation

In descriptions of new technology, *rich* is probably one of the most overused words. However, I can't think of a better term to convey the principle behind WPF. Our goal was to create a superset of features from all existing presentation technologies—from basic things like vector graphics, gradients, and bitmap effects, to more advanced things like 3D, animation, media, and typography. The other key part of the principle was the word *platform.* The goal was to create not merely a runtime player for rich content, but rather an application platform that people could use to build large-scale applications and even extend the platform to do new things that we never envisioned.

Build a Programmable Platform

Early on, the WPF team decided that both a markup (declarative) and code (imperative) programming model were needed for the platform. As we looked around at the time, it became clear that developers were embracing the new managed-code environments. Quickly, the principle of a programmable platform became a principle of a managed programming model. The goal was to make managed code the native programming model of the system, not a tacked-on layer.

Build a Declarative Platform

From the perspective of both customers and software developers, it seemed clear that the industry was moving to a more and more declarative programming model. We knew that for WPF to be successful, we needed a rich, consistent, and complete markup-based programming model. Again, a look at what was going on in the industry made it clear that XML was becoming the de facto standard for data interchange, so we decided to build an XML programming model, which became XAML (Extensible Application Markup Language).

Integrate UI, Documents, and Media

Probably the biggest problem facing customers who were building applications was the separation of pieces of functionality into isolated islands. There was one platform for building user interfaces, another for building a document, and a host of platforms for building media, depending on what the medium was (3D, 2D, video, animation, etc.). Before embarking on building a new presentation system, we set a hard-and-fast goal: The integration of UI, documents, and media would be the top priority for the entire team.

Incorporate the Best of the Web, and the Best of Windows

The goal here was to take the best features from the last 20 years of Windows development and the best features from the last 10 years of Web development and create a new platform. The Web offers a great simple markup model, deployment model, common frame for applications, and rich server connectivity. Windows offers a rich client model, simple programming model, control over the look and feel of an application, and rich networking services. The challenge was to blur the line between Web applications and Windows applications.

Integrate Developers and Designers

As applications become graphically richer and cater more to user experience, an entirely new community must be integrated into the development process. Media companies (print, online, television, etc.) have long known that a variety of designer roles need to be filled to create a great experience for customers, and now we are seeing that same requirement for software applications. Historically the tools that designers used were completely disconnected from the software construction process: Designers used tools like Adobe Photoshop or Adobe Illustrator to create rich designs, only to have developers balk when they tried to implement them. Creating a unified system that could natively support the features that designers required, and using a markup format (XAML) that would allow for seamless interoperability between tools, were two of the outcomes of this principle.

About This Book

Many books on WPF are, and will be, available. When I first thought of writing a book, I wanted to make sure that mine would offer something unique. This book is designed for application developers; it is intended as a conceptual reference book covering most of WPF.

I chose each word in the preceding statement carefully.

This book is about applications. There are really two types of software: software designed to communicate with people, and software designed to communicate with software. I use the term *application* to mean software written primarily for communication with people. Fundamentally, WPF is all about communication with people.

This is a book for *developers.* I wanted to present a very code-centric view of the platform. I'm a developer first and foremost, and in working as an architect on the WPF team I have always considered the external developer as my number one customer. This book focuses on topics primarily for the application developer. Although a control developer will also find a lot of useful information in this book, its purpose is not to present a guide for building custom controls.

This book is about *concepts,* not just APIs. If you want an API reference, use Google or MSN search features and browse the MSDN documentation. I want to raise the abstraction and present the hows and whys of the platform design and show how the various APIs of the platform work together to add value to developers.

This book is a reference; it is organized by technical topics so that you can flip back to a section later or flip forward to a section to answer a question. You do not need to read the book from cover to cover to gain value from it.

This book covers most of WPF, not all of it. When I started writing the book, Chris Sells gave me an important piece of advice: "What you leave out is as important as what you include." Because WPF is an immense platform, to present the big picture I had to omit parts of it. This book represents what I believe are the best landmarks from which to explore the platform.

My goal with this book is to provide a map of the core concepts, how they relate to each other, and what motivated their design. I hope you'll come away from this book with a broad understanding of WPF and be able to explore the depth of the platform yourself.

Prerequisites

Before reading this book, you should be familiar with .NET. You don't need to be an expert, but you should be familiar with the basics of classes, methods, and events. The book uses only C# code in its examples. WPF is equally accessible in any .NET language; however, C# is what I use primarily for my development.

Organization

This book is organized into eight chapters and a three-part appendix. My goal was to tell the story of the WPF platform in as few chapters as possible.

- **Introduction** (Chapter 1) briefly introduces the platform and explains how the seven major components of WPF fit together. This chapter also serves as a quick start for building applications with WPF, showing how to use the SDK tools and find content in the documentation.
- **Applications** (Chapter 2) covers the structure of applications built using WPF, as well as the application services and top-level objects used by applications.
- **Controls** (Chapter 3) covers both the major design patterns in WPF controls and the major control families in WPF. Controls are the fundamental building blocks of user interfaces in WPF; if you read only one chapter in the book, this is the one.
- **Layout** (Chapter 4) covers the design of the layout system, and an overview of the six stock layout panels that ship in WPF.
- **Visuals** (Chapter 5), provides an overview of the huge surface area that is the WPF visual system. The chapter covers typography, 2D and 3D graphics, animation, video, and audio.
- **Data** (Chapter 6) covers the basics of data sources, data binding, resources, and data transfer operations.
- **Actions** (Chapter 7) provides an overview of how events, commands, and triggers work to make things happen in your application.

- **Styles** (Chapter 8) covers the styling system in WPF. Styling enables the clean separation of the designer and developer by allowing a loose coupling between the visual appearance of a UI and the programmatic structure.
- The appendix, **Base Services,** drills down into some of the low-level services in WPF. Topics covered include threading model, the property and event system, input, composition, and printing.

Acknowledgments

This book has been a massive undertaking for me. I've worked on articles, presentations, and white papers before, but nothing prepared me for the sheer volume of work it takes to condense a platform the size of WPF into a relatively short book.

I've dedicated this book to my wife, Megan. She has been constantly supportive of this project (even when I brought a laptop on numerous vacations!) and everything else I do.

The entire Avalon team has been a huge help in the creation of this book (and the product!). My manager, Ian Ellison-Taylor, supported my working on this project. Sam Bent, Jeff Bogdan, Vivek Dalvi, Namita Gupta, Mike Hillberg, Robert Ingebretsen, David Jenni, Lauren Lavoie, Ashraf Michail, Kevin Moore, Greg Schechter—the team members who helped are too many to list. I thoroughly enjoyed working with everyone on the team.

I am grateful to Don Box for pushing me to write the book, and to Chris Sells for giving me sage advice even while we were creating competing books.

My developmental editor, Michael Weinhardt, deserves a huge amount of credit for the quality of this book. Michael read, reread, edited, and re-edited every section of this book. He pushed me to never accept anything that isn't great. All the errors and bad transitions in the book are purely my fault.

Joan Murray, Karen Gettman, Julie Nahil, and the entire staff at Addison-Wesley, have done an amazing job dealing with me on this book. Stephanie Hiebert, my copy editor, spent countless hours pouring over my poor spelling, grammar, and prose, turning my ramblings into the English language.

Finally, I want to thank the technical reviewers of this book. Erick Ellis, Joe Flanigan, Jessica Fosler, Christophe Nasarre, Nick Paldino, Chris Sells, and a host of others provided great feedback. Jessica gave me some of the deepest and most constructively critical feedback that I've ever received.

I'm sure I'm forgetting many other people, and for that I apologize.

Chris Anderson
November 2006
simplegeek.com