Foreword

The late 1990s brought us the revolution of the Internet. After 15 years of moving from a server-based model of computing to a client/server-based model, the pendulum swung back heavily toward the server with the rapid growth of Web pages, HTML, and server-based applications.

There was much to like about Web applications. Designers liked them because they had lots of great ways to apply nice-looking style sheets and layouts. Companies liked them because they did away with all the expensive and risky aspects of deploying client applications. All that had to be done was to install the application on a Web server, and you were done. No risk of breaking other applications or need to physically install the software on every machine in the organization. And for document viewing, HTML was a relatively easy language to learn, so it allowed many people to do some manner of software development with no prior skills.

But not everything was perfect. Large-scale Web applications were difficult to write and manage. There were differences between browsers. There weren't very good tools for debugging and developments. The applications weren't taking advantage of all the power on the client machines—hard drives, video cards, and CPUs. And most importantly, the user interfaces generally were well-suited only to the most basic data entry. If you needed real-time display or advanced visualization, things got very difficult.

In early 2002, Windows Forms was released as part of the Microsoft .NET Framework, Version 1.0. This changed the landscape in two fundamental ways. First, it gave programmers a consistent, approachable API and toolset with which to build very sophisticated applications for Microsoft Windows without having to know the Win32 SDK forward and backward. And second, the .NET Framework and Common Language Runtime (CLR) allowed client applications to be deployed via a Web server. Once you got the .NET Framework installed on the client machines you could have true zero-cost or "no-touch" deployment.

In conjunction with this, organizations were beginning to recognize the aforementioned shortcomings of Web applications in certain scenarios, and started to once again deploy client applications.

With the release of Version 2.0 of the Microsoft .NET Framework, even more client momentum is building. Windows Forms now allows customers to build applications with the look and feel of not only Windows itself, but of Microsoft Office as well. And then they can deploy those applications using a much-improved deployment technology called ClickOnce that is integrated directly into the Microsoft Visual Studio 2005 design experience. Gone are the days when organizations had to default to writing Web applications. Now they can choose the technology that is appropriate for the task at hand, which means they can implement their vision without compromising the user experience. Version 1.0 of Windows Forms and the .NET Framework were a good start, but we Version 2.0 takes smart client development to the next level!

Matthew MacDonald understands this and has built a great resource for developers using Windows Forms to create great, rich applications. Whether the goal is to write components for internal use or a full application, this book will help you get there and deliver great results. Welcome back to the client.

Before Windows Forms, there were application developers and there were control developers. Even with Visual Basic, controls were usually authored in another language like Visual C++ and required a specific set of skills. However, with an object-oriented framework like Windows Forms, customizing control behavior is done with the same techniques as other application development, which gives developers a powerful new tool to really make their client applications deliver a great user experience that just can't be matched anywhere else. *Pro .NET 2.0 Windows Forms and Custom Controls* does an excellent job of highlighting those possibilities and equipping developers with the techniques to make them a reality. Whether it's creating an owner-drawn TreeView, using the new layout features to build dynamic interfaces, or creating skinned custom controls, this book shows you how.

The practical, task-based approach of *Pro .NET 2.0 Windows Forms and Custom Controls* allows it to cover a wide range of Windows Forms topics, but still provide the technical depth to help developers deliver features. While many other resources read more like technical reference docs, *Pro .NET 2.0 Windows Forms and Custom Controls* does an excellent job of filtering the information down to what developers really need to harness the power and innovations of Windows Forms 2.0 to deliver truly world-class client applications.

Shawn Burke
Development Manager, Windows Forms Team
Microsoft Corporation

Introduction

Four years after the .NET Framework first hit the programming scene, smart client applications still refuse to die.

This is significant because when .NET first appeared, all too many people assumed it was about to usher in a new world of Web-only programming. In fact, for a short time Microsoft's own Web site described the .NET Framework in a single sentence as a "platform for building Web services and Web applications"—ignoring the Windows technology that made the company famous.

Now that the dust has settled, it's clear that Web and Windows applications aren't locked in the final rounds of a life-or-death battle. Instead, both technologies are flourishing. And not only are both technologies gaining strength, but they're also stealing some of each other's best features. For example, the latest release of .NET gives Web developers rich controls like menus and trees that were previously the exclusive domain of Windows coders (or Web-heads who weren't afraid to write a mess of hardcore client-side JavaScript). On the other hand, Windows applications are gaining easy Web-based deployment, more-flexible layout options, and the ability to display HTML. All of these innovations point to many productive years ahead for Web and Windows developers alike.

If you've picked up this book, you've already decided to learn more about programming Windows smart clients with .NET. Although both Web and Windows applications have their strengths and weaknesses, only Windows applications allow you to break out of the confines of the browser and take full advantage of the client computer. With Windows Forms, you can play sound and video, display dynamic graphics, react to the user's actions instantaneously, and build sophisticated windowed interfaces.

In this book, you'll learn how to use all of these techniques to design state-of-the-art application interfaces. Best of all, you won't just learn how to use the existing controls of the .NET Framework—you'll also learn everything you need to extend, enhance, and customize them.

About This Book

This book focuses relentlessly on *Windows Forms*, the .NET toolkit for building modern Windows interfaces.

In this book you'll learn about several sides of user interface programming. Some of the key themes include the following:

- **Dissecting the .NET controls.** Although this book is not a reference, it contains an exhaustive tour of just about every .NET user interface element you'll ever want to use.
- **Best practices and design tips.** As a developer, you need to know more than how to add a control to a window. You also need to know how to create an entire user interface framework that's scalable, flexible, and reusable.

- **How to enhance .NET controls and build your own.** In this book, you'll learn key techniques to extend existing controls and create your own from scratch. You'll even learn how to draw controls from scratch with GDI+, the remarkable .NET drawing framework.
- **How to design elegant user interfaces for the average user.** This subject isn't the focus of the book, but you'll get a great overview from Appendix A. You'll also learn more from tips and notes throughout the book.
- **Advanced user interface techniques.** Features are neat, but how do you use them? In this book you'll see practical examples of common techniques like document-view architecture, validation, and hit testing. You'll also learn how to dynamically generate forms from a database, unshackle data binding, and build an integrated help system.

Of course, it's just as important to point out what this book *doesn't* contain. You won't find the following subjects in this book:

- **A description of core .NET concepts.** These key concepts, like namespaces, assemblies, exception handling, and metadata, are explained in countless books, including a number of excellent C# and VB .NET titles from Apress.
- **A primer on object-oriented design.** No .NET programmer can progress very far without a solid understanding of classes, interfaces, and other .NET types. In this book, many examples rely on these basics, using objects to encapsulate, organize, and transfer information.
- **A reference for Visual Studio 2005.** The new integrated design environment provides powerful customization, automation, and productivity features that deserve a book of their own. Though this book assumes you're using Visual Studio, and occasionally points out an often-overlooked feature, it assumes that you already know your way around the development environment.

You'll get the most out of this book if you've already read another, more general .NET book. If you haven't learned the .NET fundamentals yet, you'll still be able to work through this book, but you'll need to travel at a slower pace and you may need to refer to the MSDN Help files to clear up issues you'll encounter along the way.

Note This book is targeted at experienced developers who want to get the most out of .NET. If you have never programmed with a language like Visual Basic, C++/C#, or Java before, this isn't the place to start. Instead, start with an introductory book on object-oriented design or programming fundamentals. On the other hand, if you already have some experience with .NET 1.0 or 1.1, welcome—you'll find yourself right at home!

Chapter Overview

The following overview describes what each chapter covers. If you already have some experience with Windows Forms, feel free to skip from chapter to chapter. If you're relatively new to Windows Forms development, it's probably best to read through the book to make sure you learn the basics before tackling more-advanced topics.

Part I: Windows Forms Fundamentals

In this part you'll consider the core topics you need to understand to design smart clients. In Chapter 1 you'll start out by exploring the class model that underpins Windows Forms user interfaces. In Chapters 2 and 3 you'll explore the fundamental Control and Form classes. Chapter 4 describes the most common Windows controls. Chapter 5 shows how you can embed images and other binary resources into your compiled applications. Chapter 6 considers trees and lists, a hallmark of modern Windows applications. Finally, Chapters 7 and 8 consider two impressive higher-level features that are built into the Windows Forms model—GDI+ (for hand-drawing controls) and data binding (for displaying and updating data without writing tedious code).

Part II: Custom Controls

In this part, you'll tackle one of the most important areas of Windows Forms design—creating customized controls that add new features, use fine-tuned graphics, and encompass low-level details with higher-level object models. In Chapter 9 you'll learn about the basic types of custom controls you can create and see how to set up a custom control project. You'll then continue to create user controls, which combine other controls into reusable groups (Chapter 10); derived controls, which enhance existing .NET control classes (Chapter 11); and owner-drawn controls, which use GDI+ to render UI from scratch (Chapter 12). Chapter 13 shows how you can add design-time support so your custom controls behave properly at design time.

Part III: Modern Controls

In this part, you'll branch out to some of the most powerful Windows Forms controls. In Chapter 14, you'll explore the new ToolStrip, which provides a thoroughly customizable and flexible model for toolbars, menus, and status bars. In Chapter 15 you'll consider the DataGridView—an all-in-one grid control for displaying data. In Chapter 16 you'll look at the still woefully weak support for sound and video in the .NET Framework, and learn how to improve the picture with interop. Finally, in Chapter 17 you'll learn how the WebBrowser lets you show HTML pages in a Windows application, and you'll learn some remarkable tricks for integrating the two (with Windows code that manipulates the page and JavaScript Web code that triggers actions in your application).

Part IV: Windows Forms Techniques

In this part, you'll considerable indispensable techniques for serious Windows Forms programmers. In Chapter 18 you'll consider a host of approaches to validation, from masked edit controls to custom validation components that mimic ASP.NET, and perform their work automatically. Chapter 19 tackles MDI and SDI interfaces and shows you how to build a document-view framework. Chapter 20 explores the world of multithreading, and provides practical advice on how to write safe, performance-asynchronous code in a Windows application. Chapter 21 shows how you can build a new breed of Windows application with the highly adaptable, Web-like layout engines. Chapter 22 considers how you can build Help and integrate it into your application.

Part V: Advanced Custom Controls

The final part considers some advanced topics that illustrate interesting subjects and help you extend your expertise. In Chapter 23 you'll see how to build slick applications with shaped forms, skinned controls, and custom buttons. In Chapter 24 you'll see a complete vector-drawing application that contrasts custom controls against a more powerful drawing model. Chapter 25 considers how you can extend existing controls with custom extender providers, and Chapter 26 picks up where Chapter 17 left off, by exploring more features and frills of design-time support for custom controls.

Appendixes

In the appendixes, you'll take a look at principles for user interface design in any language (Appendix A) and the new ClickOnce deployment technology (Appendix B).

Moving from .NET 1.x to .NET 2.0

If you've programmed with .NET 1.x, you'll find that a great deal remains the same in .NET 2.0. The underlying model for creating Windows Forms applications and custom controls remains unchanged. However, there are some significant new feature areas.

For the most part, this book doesn't emphasize the difference between features that have existed since .NET 1.x and those that are new in .NET 2.0, chiefly because some significant features and programming techniques have remained the same since .NET 1.0, but are still misunderstood by many developers. However, if you have extensive .NET 1.x programming experience, you may want to begin by exploring some of the feature areas that have changed the most.

The following list of the 14 most important changes points you to the right chapters:

1. **The SplitContainer control (Chapter 3).** Finally, there's an easier way to design complex windows with multiple split panes. It's a small addition, but it's a major convenience.

2. **AutoComplete (Chapter 4).** You see it in lists and text boxes throughout the Windows world. Now there's an easy way to get AutoComplete behavior without coding it by hand.

3. **Design-time support for resources (Chapter 5).** Deploying image files with your application is too fragile. But in the past, the alternative (embedding them in an assembly) has been awkward. Visual Studio 2005 solves the problem with new features for embedding and managing resources.

4. **Visual styles (Chapter 7).** Not only does .NET 2.0 make it easy to take advantage of Windows XP visual styles (for all controls), it also includes a new set of classes that lets you paint custom controls using the Windows XP theming API.

5. **Automatic data binding (Chapter 8).** Some love it; some hate it. Either way, you'll need to understand quite a bit about the new support for code-free data binding if you want to have any chance of creating a practical, scalable application.

6. **The ToolStrip control (Chapter 14).** Microsoft solves the problems of the out-of-date menu, status bar, and toolbar in one step with a new model revolving around the ToolStrip class. Best of all, the ToolStrip is endlessly customizable.

7. **The DataGridView control (Chapter 15).** The underpowered and inflexible DataGrid of .NET 1.x fame is replaced with a completely new grid control. Highlights include a fine-grained style model and support for extremely large sets of data through virtualization.

8. **The SoundPlayer control (Chapter 16).** This new control gives basic WAV playback features, but it still comes up far short, with no support for more-modern standards like MP3 audio or video. (Chapter 16 also shows you how to get around these problems with the Quartz library.)

9. **The WebBrowser control (Chapter 17).** Finally, a clean, easy way to show a Web page in a window. Use it with local or remote data. Best of all, you have the ability to explore the DOM model of your page, and react to JavaScript events in your Windows code.

10. **Masked editing (Chapter 18).** A new MaskedEdit control gives you a text box with masked editing features. You can also use lower-level classes to integrate masked editing into any control.

11. **The BackgroundWorker component (Chapter 20).** Use this class to perform an asynchronous task without worrying about marshalling your code to the user-interface thread. (However, though the BackgroundWorker fits certain scenarios, you'll still need to take control of multithreading on your own for many tasks.)

12. **Dynamic interfaces (Chapter 21).** It just might be the most underreported yet most significant shift in Windows applications. The new layout managers allow you to build flowing, Web-like applications that lay out different modules in a variety of flexible ways. They also make it easier to deal with expanding and contracting text in localization scenarios.

13. **Smart tags (Chapter 26).** Smart tags provide a helpful panel through which you perform a variety of tasks with a control at design time. Why not build your own for custom controls?

14. **ClickOnce (Appendix B).** ClickOnce doesn't really change the existing .NET deployment model—instead, it adds a higher-level set of features you can use to easily support self-updating applications, particularly over the Web or an intranet.

This list doesn't include all the minor features and tune-ups you'll discover as you explore Windows Forms and read through this book.

What's Still Missing in .NET 2.0

Even though .NET 2.0 is more than a minor upgrade to .NET 1.x, there is still a host of features that longtime Windows developers may find lacking.

Here are some examples of what you still *won't* find:

- Window management, including tabbed and dockable windows.
- Charting and other controls for data visualization.

- A commanding architecture (so that multiple actions in a user interface trigger the same operation).
- Markup-based layout features.
- Support for MS Help 2.0, the (unsupported) standard that's used for the Visual Studio help files.
- A document-view framework for building applications.
- More high-level controls (like an Outlook bar, task panes, a wizard framework, and so on).

Some of these features are easy to develop on your own, while others are extremely difficult to do properly. In all these cases, third-party components have already emerged to fill the gaps (with varying levels of success). However, it's unlikely than a native Framework solution will emerge for any of these features, because the focus in rich client development is shifting to the new Avalon framework, which is a part of the upcoming Windows Vista operating system.

Note Some third-party-component developers that you might want to check out are `www.dotnetmagic.com`, `www.divil.co.uk`, and `www.actiprosoftware.com`.

Conventions Used in this Book

You know the drill. This book uses *italics* to emphasize new terms and concepts. Blocks of code use `constant width formatting`. Note and tip boxes are scattered throughout the book to identify special considerations and useful tricks you might want to use.

Code Samples

It's a good idea to download the most recent, up-to-date code samples. You'll need to do this to test most of the more-sophisticated code examples described in this book, because the less-important details are usually left out. Instead, this book focuses on the most important sections so that you don't need to wade through needless extra pages to understand an important concept. To download the source code, navigate to `www.prosetech.com`. The source code for this book is also available to readers at `http://www.apress.com` in the Source Code section. On the Apress Web site, you can also check for errata and find related titles from Apress.

Variable Naming

Hungarian notation, which names variables according to their data type (like strFirstName instead of FirstName), was the preferred standard for C++ and Visual Basic 6. These days, Hungarian notation is showing its age. In the world of .NET, where memory management is handled automatically, it seems a little backward to refer to a variable by its data type, especially when the data type may change without any serious consequences, and the majority of variables

are storing references to full-fledged objects. Microsoft now steers clear of variable prefixes, and recommends using simple names.

In this book, data-type prefixes aren't used for variables. The only significant exception is with control variables, where it is still a useful trick to distinguish between types of controls (like txtUserName and lstUserCountry), and with some data objects. Of course, when you create your own programs you're free to follow whatever variable naming convention you prefer, provided you make the effort to adopt complete consistency across all your projects (and ideally across all the projects in your organization).

Note Microsoft provides detailed information about recommended coding and naming standards in the MSDN (see `http://msdn.microsoft.com/library/en-us/cpgenref/html/cpconNETFrameworkDesignGuidelines.asp`). If you plan to release a component for use by third-party developers, you'll need to read these documents carefully.

Feedback

This book has the ambitious goal of being the best tutorial and reference for programming Windows Forms. Toward that end, your comments and suggestions are extremely helpful. You can send complaints, adulation, and everything in between directly to `apress@prosetech.com`. I can't solve your .NET problems or critique your code, but I will benefit from information about what this book did right and wrong (and what it may have done in an utterly confusing way).