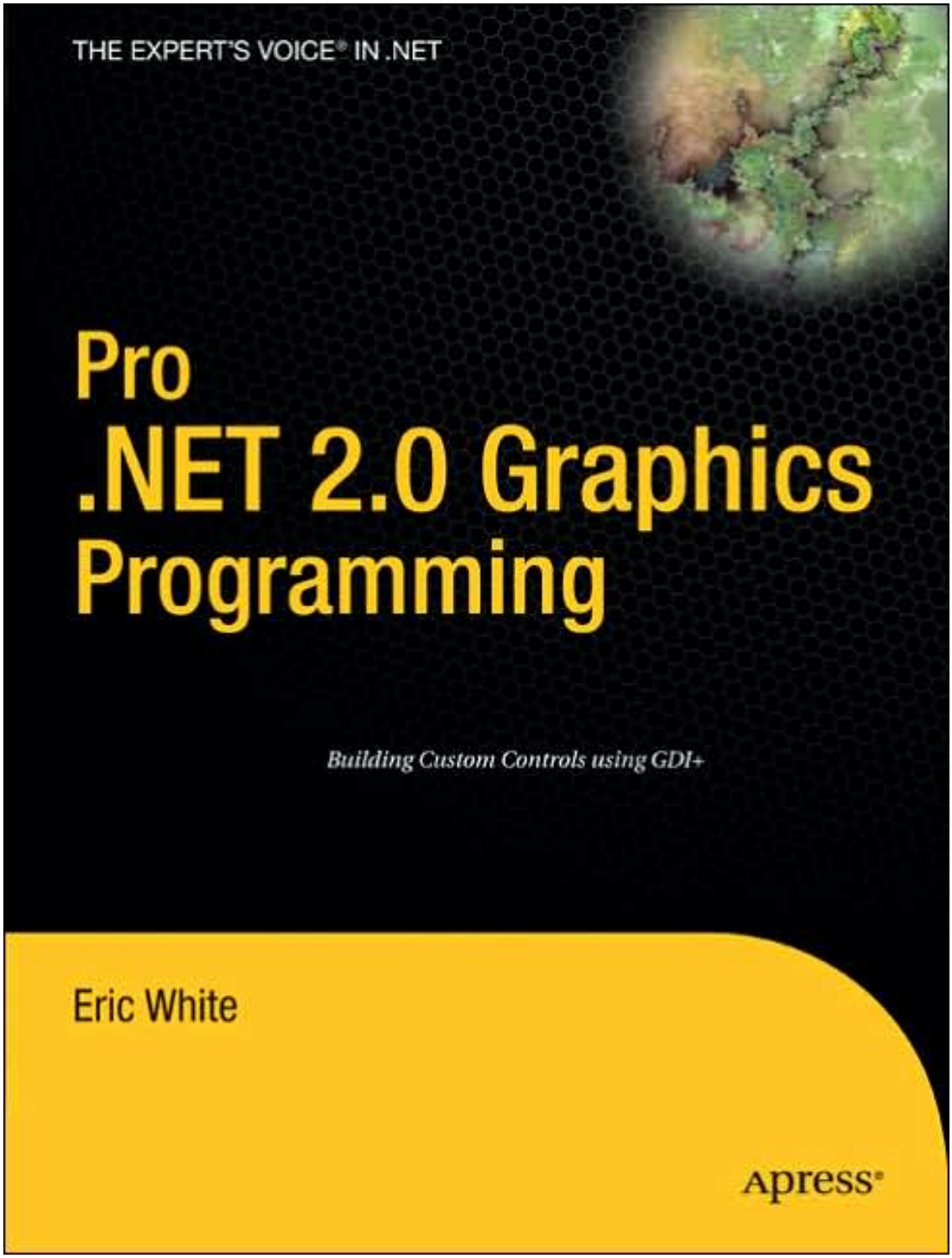
THE EXPERT'S VOICE® IN .NET

Pro .NET 2.0 Graphics Programming

Building Custom Controls using GDI+

Eric White

Apress®



Pro .NET 2.0 Graphics Programming

Eric White

Apress © 2005 (472 pages)
ISBN:1590594452

Whether you are using Windows Forms to build rich-client business applications, or the .NET framework to build powerful web applications, this comprehensive guide aims to provide you with all the information you need to build effective custom controls.

Table of Contents

Pro .NET 2.0 Graphics Programming

by Eric White

Apress © 2005 (472 pages)

ISBN:1590594452

Whether you are using Windows Forms to build rich-client business applications, or the .NET framework to build powerful web applications, this comprehensive guide aims to provide you with all the information you need to build effective custom controls.

Back Cover

Whether you are using Windows Forms to build rich-client business applications, or the ASP.NET 2.0 framework to build powerful web applications or web services, the use of well-designed graphics will greatly enhance their usability, impact, and visual appeal. This book provides a comprehensive guide to the use of graphics in .NET applications and aims to provide you with all the information you need to build effective custom controls.

The opening section of the book investigates the .NET Framework classes that implement graphics. It covers all of the classes, methods, and techniques needed to create, manipulate, and display precise graphics in a form, a page being sent to a printer, or an image.

On this foundation, the second section describes how to design and build effective custom controls for use in a business environment. Topics covered include building composite controls, implementing keyboard navigation, and enhancing design-time support.

The final section of the book explores the use of GDI+ and ASP.NET to build custom controls that can provide reusable, GUI components for web projects, and to deliver customized graphics over the Internet.

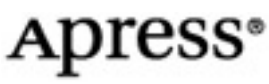
About the Author

Eric White is an independent software consultant with over twenty years of experience building management information systems, accounting systems, and other types of rich-client and n -tier database applications. He has written custom controls in numerous windowing systems, including all versions of Microsoft Windows since 2.0, Macintosh, OS/2, X/Motif, and others. White has particular interest in object-oriented design methodologies, including use case analysis, UML, and design patterns. After years of working with too many varieties of technologies to list, he currently specializes in C#, Visual Basic .NET, ASP.NET, ADO.NET, XML, COM+, GDI+, SQL Server, and other Microsoft technologies.

Next

Pro .NET 2.0 Graphics Programming

Eric White



Apress

ISBN (pbk): 1-59059-445-2

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Ewan Buckingham

Technical Reviewer: Mark Horner

Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis, Jason Gilmore, Jonathan Hassell, Chris Mills, Dominic Shakeshaft, Jim Sumser

Project Manager: Beth Christmas

Copy Edit Manager: Nicole LeClerc

Copy Editor: Marilyn Smith

Assistant Production Director: Kari Brooks-Copony

Production Editor: Kelly Winquist

Compositor: Kinetic Publishing Services, LLC

Proofreader: Linda Seifert

Indexer: John Collin

Artist: Kinetic Publishing Services, LLC

Interior Designer: Diana Van Winkle, Van Winkle Design Group

Cover Designer: Kurt Krames

Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit http://www.springeronline.com.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930,

fax 510-549-5939, e-mail info@apress.com, or visit http://www.apress.com.

The source code for this book is available to readers at http://www.apress.com in the Source Code section. You will need to answer questions pertaining to this book in order to successfully download the code.

I dedicate this book to the memory of my parents, David and Thelma White.

I also dedicate this book to Autumn (with the freckle in her eye).

About the Author

ERIC WHITE is an independent software consultant with more than 20 years of experience building management information systems, accounting systems, and other types of rich-client and *n*-tier database applications. He has written custom controls in numerous windowing systems—all versions of Microsoft Windows since 2.0, Macintosh, OS/2, X/Motif, and others. Eric is particularly interested in object-oriented design methodologies, including use case analysis, UML, and design patterns. After years of working with too many varieties of technologies to list, he currently specializes in C#, Visual Basic .NET, ASP.NET, ADO.NET, GDI+, XML, COM+, and SQL Server.

About the Technical Reviewer

MARK HORNER is a principal enterprise architect and .NET application consultant with Torville Software. He is a 25-year veteran of the industry and has worked with a host of blue-chip organizations, including Citibank, NRMA Insurance, ANZ Banking Group, Unilever, Hewlett-Packard, British Aerospace, and CPA Australia. Mark has authored five books, including *Pro .NET 2.0 Code and Design Standards in C#* (Apress, 2005).

Acknowledgments

Writing a book requires the efforts of many people besides the author. Therefore, I would like to acknowledge the efforts of the following Apress staff: Ewan Buckingham, Beth Christmas, and Marilyn Smith. I would also like to thank the technical reviewer, Mark Horner, for his insight and advocacy for the reader.

Next

Previous Next

Introduction

Overview

The release of the .NET Framework has changed the programming landscape for Microsoft application developers. Graphics programming has been affected as much as any other area. Whereas the Windows Graphical Device Interface (GDI) was once used to draw to a device, the .NET Framework now gives us *GDI+*.

GDI+ consists of a number of namespaces and classes, made available as part of the .NET Framework and designed specifically to allow developers to do custom drawing in web and Windows controls. Effectively acting as a wrapper around the old GDI, GDI+ provides an easy-to-understand, inheritance-based object model for use by .NET programmers.

The increased level of usability of GDI+ may draw many more programmers into custom drawing. Those from a C++/Microsoft Foundation Classes (MFC) background will already be familiar with many of the approaches in GDI+. Programmers who worked with Visual Basic will find in GDI+ an accessible way to develop controls that augment those shipped with the .NET Framework.

A substantial part of the .NET Framework base-class library falls under the GDI+ banner. This book concentrates on the areas and features of GDI+ that are relevant to programmers who are most likely to work with these classes; that is, programmers who want to develop *custom controls*.

Previous Next

Previous Next

Who Is This Book For?

This book is designed to appeal to any developer who wishes to appreciate and exploit the functionality of the GDI+ base classes provided as part of the .NET Framework. In particular, the book is aimed at those working in a business-application environment, writing GDI+ code for Windows Forms in order to create custom controls.

The examples in this book are presented in the C# language. They should be accessible to both intermediate-level C# programmers and relatively experienced developers capable of recognizing the standard constructs and features of an object-oriented language.

However, the choice of C# for the examples in this book is less important than the fact that it is a book about developing with GDI+ and custom controls in .NET. Indeed, the GDI+ techniques covered here are language-neutral. They can be applied, with the appropriate language knowledge, using other .NET Framework languages such as Visual Basic .NET and Visual C++ .NET.

Previous Next

Previous Next

What Does This Book Cover?

In this book, I'll cover GDI+ in three broad steps. First, I introduce GDI+ and its main features. Next, I tackle the nuts and bolts of GDI+ programming—the classes, methods, properties, and events that are most important to the application of GDI+ in custom controls, with plenty of description and demonstration. Then the chapters focus specifically on custom control architecture and development. I'll cover the issues you need to consider when deciding which features to build into your application, taking into account the requirements of the user interface.

Here is a summary of each chapter's contents:

- Chapter 1: Here, I provide an overview of GDI+ and give some context for the development and use of custom controls in .NET applications.
- Chapter 2: This chapter looks at GDI+ fundamentals such as drawing surfaces and coordinate systems. It includes an introduction to the `Graphics` object, which encapsulates a drawing surface. You'll see that there are basically three types of drawing surfaces: screens, bitmaps, and printers. The `Graphics` object provides methods for drawing and painting onto your drawing surface, as well as for managing many other features; for example, measuring units, transformations, clipping, and invalidation. Hence, the `Graphics` class provides the foundation for much of the rest of the book.
- Chapter 3: This chapter covers pens and brushes. GDI+ makes a distinction between drawing and filling, and this fact is encapsulated well in the existence and juxtaposition of the `Pen` and `Brush` classes. These two classes are fundamental graphics tools, and you'll explore their versatility in this chapter.
- Chapter 4: This chapter is about text and fonts. You'll see how GDI+ gives you programmatic control over the content and nature of the text in your graphics. You'll examine the `Font` and `FontFamily` classes, and understand how they relate to the rest of your graphics.
- Chapter 5: The focus of this chapter is images and image manipulation. You'll explore the two GDI+ classes that allow you to handle images: `Image` and `Bitmap`, and even learn how to create and save images.
- Chapter 6: This chapter introduces the important subjects of paths and regions, encapsulated in GDI+ by the `GraphicsPath` and `Region` classes. You'll see how these classes further highlight the distinction between drawing outlines and filling areas, enabling you to work with irregular shapes and to group sets of shapes together for more effective processing.
- Chapter 7: This chapter expands on the use of the `GraphicsPath` and `Region` classes. You'll see how you can use these objects in operations such as clipping and invalidation, to create some useful and powerful effects.
- Chapter 8: This chapter explains how to handle transformations in GDI+. Although translations, rotations, and shearing aren't commonly used in custom control development, this chapter will highlight some interesting techniques that may be appropriate in some applications.
- Chapter 9: This chapter turns to the subject of printing. You'll develop an appreciation of how to control differences between what you see on the screen and what the printer outputs to the page.
- Chapter 10: This chapter describes how to set up an alternative coordinate system, which you may find helpful in your development of custom controls. It involves subtle changes in the semantics of the `Graphics` class. You cannot inherit from the `Graphics` class itself (because it is defined as `sealed`), but you can "modify" its behavior by employing a design pattern called the decorator pattern, in which you define your own class whose methods have identical signatures to the methods in the `Graphics` class.
- Chapter 11: This chapter provides in-depth coverage of custom controls, including their characteristics and how they compare with components. You'll see examples of custom controls that demonstrate their properties, focus, and events. You'll also learn how to derive a custom control from an existing Windows Forms control and how to develop composite custom controls.
- Chapter 12: This chapter considers some of the ways to build in functionality for your custom control to make it easier for other

developers to use your control in their code. It focuses on better integrating your controls with Visual Studio .NET, such as creating modal boxes and drop-down boxes to edit properties. All these features are intended to make life easier, at design time, for the developer using your control.

- Chapter 13: This chapter looks at your options for implementing scrolling in your custom controls. You'll learn about the built-in support for scrolling in the Windows Forms classes, and then why it is preferable to implement scrolling through a custom control. You'll also discover how to implement smooth scrolling, to add a more polished feel to your application.

- Chapter 14: This chapter is about another user-interaction feature: the mouse. You'll see how to develop your control so that it performs different functions when the mouse is used in different ways, including hit testing, mouse-event routing, dragging, and drag-scrolling. By their nature, graphical user interfaces are visual, so you'll also take a look at cursors in your applications and what you can do with them.

At the back of the book, you'll find three appendixes with additional information related to working with GDI+ and the examples in this book. Appendix A covers dealing with assemblies and namespaces. Appendix B is about working with the console in Windows application. Finally, Appendix C looks at disposing of objects based on classes that implement the `IDisposable` interface.

Previous Next

Previous Next

What You Need to Run the Examples

To run the examples in this book, you'll need the following:

- A suitable operating system (at the time of writing, Microsoft recommends Windows XP Professional)
- The .NET Framework Software Development Kit (SDK)

The examples in this book are developed using the Visual Studio .NET integrated development environment (IDE). It is possible to build the examples *without* this IDE, but I don't cover that in the book.

I recommend that you also download the complete source code for the examples in this book, from www.apress.com, as described next. Whether you plan to cut and paste the code from these source files or type the code for yourself, the source code provides a valuable way to check for errors in your code.

Previous Next

Previous Next

Source Code and Errata

As you work through the examples in this book, you may choose either to type in all the code by hand or to use the source code that accompanies the book. Many readers prefer the former, because it's a good way to become familiar with the coding techniques.

Whether or not you want to type in the code, it's useful to have a copy of the source code handy. If you type in the code, you can use the source code to check the results you should be getting; it should be your first stop if you think you might have entered something incorrectly. And if you don't like typing, you'll definitely need to download the source code from the web site. Either way, the source code will help you with updates and debugging.

All the source code used in this book is available for download at www.apress.com. Simply click the Source Code link on this book's web page to navigate to where you can obtain the code. The downloadable source code files have been archived using WinZip. You'll need to extract the files you've downloaded to a folder on your hard drive, using a decompression program such as WinZip or PKUnzip. When you extract the files, the code will be extracted into chapter folders. When you start the extraction process, ensure that your decompression program is set to use folder names.

The publishing team and I have made every effort to make sure that there are no errors in the text or in the code. However, if you find an error in this book, like a spelling mistake or a faulty piece of code, we would be very grateful to hear about it. By sending in errata, you will be helping us provide even higher-quality information. To submit errata, simply use the Submit Errata link on the book's page on the Apress web site. We'll check the information, and (if appropriate) we'll post a message to the errata pages, and then use it in subsequent editions of the book.

Previous Next

Previous Next

Contact the Author

Please feel free to contact the author with questions and comments regarding the book. You can reach Eric at gdiplus@ericwhite.com.

Previous Next

Previous Next

Chapter 1: .NET Graphics Programming

Overview

Writing graphics code is one of the most enjoyable tasks in the computer programming arena. You may be building a custom graphical window that presents data in a more visible, more accessible fashion. You may be interactively creating graphics that are served to the user as part of a web site. Whatever you're doing in this area, the job of writing code to create colorful, effective graphics is an immensely pleasing and satisfying one.

GDI+ is the next-generation graphics device interface for Microsoft Windows operating systems. It is the future of graphics programming for Windows. Its object-oriented class library allows you to build many types of graphics applications.

One of the most powerful ways to use GDI+ is to build custom controls. Windows Forms comes with a stock of comprehensive controls, but there will certainly come a time when you need more. You can build your own custom controls that enhance the power and usability of your application.

This book focuses on the combination of GDI+ and custom controls, and specifically on the creation of custom controls using graphics in the .NET Framework and the C# language. We'll begin by looking at the GDI+ basics, before moving on to their application in the creation of useful controls. In this chapter, we'll look at two main areas:

- What GDI+ is—its relationship to the .NET Framework, its purpose, how it relates to previous graphics libraries, and a brief introduction to the namespaces and classes it encompasses.
- What custom controls are—an overview of Windows Forms and Web Forms custom controls, and how .NET helps you to create them.

So, let's start by examining GDI+ and the .NET Framework.

Previous Next

Previous Next

GDI+ and .NET

Most of you will already be aware of the many and varied reasons for adopting .NET, and will already be familiar with C#. But as a starting point, let's highlight two benefits of the .NET platform that are particularly relevant to this book.

Client Application Development with Windows Forms

.NET offers rich client application development via Windows Forms. Windows Forms is a very cleanly designed class hierarchy for building Windows applications. A highly interactive environment with properties and events, combined with a truly object-oriented approach, makes for an exceptionally powerful toolset. When you use it with the distributed computing features in the .NET Framework, you can build n-tier applications easily. The facilities for building custom controls are the best that I have ever seen.

Web Application Development with Web Forms

.NET allows web application development via ASP.NET and Web Forms. Together, ASP.NET and Web Forms provide a wonderful environment for building web sites. Many of the ideas and technologies in ASP.NET were previously evident in other environments, but there are also some completely new ideas. Because of the way that ASP.NET is built, you avoid the debugging issues that are problematic with other similar technologies. Among all the integrated development environments (IDEs) I've seen, Visual Studio .NET has the best features for improving programmer productivity.

.NET Features

.NET supplies an advanced set of features to help you create custom controls, including the following:

- **True integration of properties:** Properties are implemented in a Visual Basic-like style, as opposed to the "magic naming convention" used to implement properties in the Java world.
- **Effective event handling:** The inclusion of delegates as an integral part of the language provides a very good mechanism for writing event handlers.
- **Better design-time environment:** Visual Studio .NET gives you an unparalleled level of power for building custom controls. Not only can you seamlessly deploy custom controls in the Design window, but you also can explicitly control the appearance and manipulation of the controls and implement property editors in a variety of ways. The inclusion of attributes in the languages allows you to configure the design-time environment.
- **Powerful graphics device interface:** As you'll see in this book, using GDI+ to build custom controls gives you a broad and colorful palette with which you can "paint" your applications.
- **Effective infrastructure for distribution of custom controls:** After you've built your custom controls, you may want to distribute them to developers in other programming groups or sell them. You may also want to purchase commercial custom controls from others. The assembly infrastructure in .NET solves the problems of previous component distribution technologies.

We've seen some of these features in other development platforms before, but until now, we've never seen them all in a single platform. Their combination provides a powerful infrastructure for building custom controls.

Previous Next

Previous Next

What is GDI+?

GDI+ is Microsoft's new .NET Framework class library for graphical programming. Because it is part of the .NET Framework, it is object-oriented, of course. Organized into six namespaces, GDI+ is a set of classes that are designed to work together. In this way, it is similar to other areas of functionality in .NET, such as ADO.NET. Let's look at some of the major features of GDI+:

- GDI+ provides three types of drawing surfaces: windows, bitmaps, and printers.
- GDI+ provides tools that allow you to draw two-dimensional "line-drawings" on any drawing surface. These include facilities for drawing lines, many types of shapes and polygons, and curves, with a vast variety of brushes and pens. You have a number of types of transformations at your disposal, allowing you to create sophisticated effects with great ease.
- The text-drawing features in GDI+ are extensive, and the anti-aliasing technology is particularly impressive. *Anti-aliasing* is a technique to approve the appearance of graphics and text. We have all seen the "jaggies," or stairsteps, when drawing diagonal lines. Anti-aliasing reduces this effect.
- GDI+ has image and bitmap support. You can read images and draw them onto any drawing surface. You can also create images and draw within them.
- GDI+ supports printing. Print preview capabilities are obtained easily, with very little additional effort on the part of the developer.
- GDI+ is designed to work with any kind of .NET application.

Since GDI+ is part of the .NET Framework, and hence available to both Windows Forms and ASP.NET applications, you can (via an appropriate class design) write your custom graphical drawing code in such a way that you can use the *same* code in *both* types of applications. If you use the .NET Framework to write web services, you can wrap the same graphical code for use in those applications, and hence (for example) serve up custom images to clients of the web services that you write.

So, when building ASP.NET applications, you can use GDI+ to create images and serve them to the client's browser. When implementing a server farm, you can load-balance by distributing a computationally intensive operation that creates graphical images over a number of computers or processors. Client applications often need to display graphics, either in custom controls or in forms. Web services can also make use of GDI+ to distribute graphical image generation among computers separated by a firewall or servers using technologies other than .NET.

In this book, we are not going to explore ASP.NET applications, nor building web services. We are going to focus solely on building Windows applications using Windows Forms.

To begin, let's place GDI+ in context and briefly consider the underlying technology: the Windows Graphical Device Interface (GDI).

Previous Next

Previous Next

GDI and GDI+

GDI+ is based on GDI, which is the part of the Windows Application Programming Interface (API) that handles graphics. In fact, GDI+ is effectively a "wrapper" around GDI (similar to the way that the Microsoft Foundation Classes, or MFC, wrapped GDI with its own classes). Perhaps someday Microsoft will implement GDI+ in a more native fashion, but for now, it is a wrapper around the Windows programming interface.

Note The term *GDI* refers to the graphics programming library that is part of the Windows API. The term *GDI+*refers to the managed code GDI+ class library that comes as part of the .NET Framework.

GDI offers a layer of abstraction, shielding the application developer from the need to program for each specific display device that the application may come across. Thus, when you want your application to draw to a screen, the application executes the appropriate GDI function, and then GDI works out how to communicate that to the video card. GDI is typically implemented through C or C++ programs (although many languages—including Visual Basic, C#, and Visual Basic .NET—can use GDI directly in much the same way that they are able to use many other parts of the Windows API).

While GDI+ sits above GDI, it offers two significant advantages. First, GDI+ adds capabilities to GDI by the inclusion of classes that implement functionality that would be difficult to write in straight GDI. To use such functionality in a GDI program, you would need to either acquire a library that supplies that functionality or write it yourself. For example, GDI+ provides specific enhancements for drawing semitransparently, access to gradient brushes, support for a wider range of image formats, easier programming of graphics paths and regions, and more powerful transformations. You'll encounter all of these during the course of this book. Second, GDI+ is fully integrated into the .NET Framework. GDI is a fairly complex, inaccessible technology. GDI+ has made the job of the programmer much easier.

The integration of GDI+ into the .NET Framework and its improved programming interface combine to bring other benefits:

- GDI+ uses an object-oriented class hierarchy. Programming GDI+ consists of creating objects, setting their properties, and calling their methods.
- GDI+ uses method overloading. Thus, when you call a method of a GDI+ object, you often have the opportunity to choose the version (or overload) of the method that takes the set of arguments most appropriate for the situation.
- GDI+ recognizes the distinction between drawing outlines and filling regions, and provides two separate operations for these tasks. However, GDI combines the two operations, and that leads to programmatic idiosyncrasies such as hollow brushes and hollow pens. If you've had experience with hollow pens and brushes in GDI, you know that they are a bit of a kluge. If you are never going to use GDI directly, and are always going to use GDI+, you don't need to worry about them; you will never see them. The result is a far cleaner programming interface.
- GDI+ hides some aspects of state management from the developer. GDI (not GDI+) keeps too much state, such as the notion of a current pen, current brush, and current point. Such artificial and unnecessary constraints are eliminated with GDI+. GDI+ shields the programmer from this complexity. This facilitates a simpler approach that eliminates the bugs that can occur when a developer doesn't completely set all necessary state (and thus "inherits" state randomly from other parts of the code).

For example, every line drawing method in GDI+ requires a `Pen` object. GDI+ knows what pen is currently part of the state of GDI, and if the pen passed as an argument to the GDI+ method is different from the pen currently selected in the device context, GDI+ automatically changes the pen in the device context. If the pen passed to the method is the same as that currently selected in the device context, GDI+ doesn't need to change the pen in the device context. GDI+ still maintains some state, such as transformations and clipping regions, but these can arguably be considered to be an attribute of the drawing surface, and should be maintained as state, whereas the notion of a current pen really is quite artificial.

GDI+ benefits from the perspective gained from the experience of building many graphical programming interfaces. It is a very clean, consistent, and powerful set of high-performance classes that provides most of the graphics functionality that you need. However, there are places where the functionality of GDI+ is limited, such as with bit block transfers or carets. Where GDI+ doesn't meet your needs, you can always drop down into the GDI API and use GDI in conjunction with GDI+. You'll see how to do this later in the book, when we discuss how to handle scrolling and mouse events in custom controls (scrolling is covered in Chapter 13, and mouse events are discussed in Chapter 14).

WHAT ABOUT DIRECTX?

DirectX is another Microsoft technology related to graphics programming. However, DirectX differs significantly from GDI+ in both purpose and programming interface. DirectX is a suite of multimedia APIs that give applications the ability to access features of PC hardware (such as sound and graphics acceleration cards), enabling them to deliver high-performance output. Such high-quality presentation is commonly encountered in PC games where three-dimensional effects are a crucial part of the application.

GDI+ and DirectX are both libraries for writing graphical applications, but in our consideration of graphics for Windows and ASP.NET business applications, we need to concern ourselves only with GDI+. For all practical purposes, and for the purposes of this book, the two technologies are mutually exclusive.

The GDI+ Namespaces

As I mentioned earlier, all of the GDI+ functionality is contained in six namespaces that are included in the .NET Framework. Table 1-1 describes the six GDI+ namespaces.

Table 1-1: GDI+ Namespaces

Namespace	Description
`System.Drawing`	Provides basic graphics functionality, including drawing surfaces, images, colors, brushes, pens, and fonts.
`System.Drawing.Drawing2D`	Provides advanced raster and vector graphics functionality.
`System.Drawing.Imaging`	Provides advanced imaging functionality, over and above that provided in the `System.Drawing` namespace.
`System.Drawing.Printing`	Provides print and print preview functionality.
`System.Drawing.Text`	Provides advanced font functionality, over and above that provided in the `System.Drawing` namespace.
`System.Drawing.Design`	Provides functionality for enhancing design-time support of custom controls. Includes classes for developing custom `UITypeEditor` classes, which allow you to customize behavior of custom controls in the Design window of Visual Studio .NET.

All of these namespaces are contained in a DLL file called `System.Drawing.dll`. You'll meet many of the classes as you progress through the early chapters of this book.

Since the focus of the book is the use of GDI+ in the construction of custom controls, let's now take a closer look at what that will require.

Previous Next

Previous Next

Custom Controls

You can use two fundamental types of custom controls in .NET:

- **Windows Forms custom controls:** These controls are used in rich client applications. These are typically highly responsive custom controls. For example, a Windows Form custom control may track the mouse when it enters the control, and it may change the mouse cursor or even the appearance of the control itself.
- **Web Forms custom controls:** These controls are used in web applications. Web Forms controls will typically be less dynamically interactive than Windows Forms custom controls.

This book will focus on Windows Forms custom controls.

Benefits of Custom Controls

Custom controls are one of the most powerful approaches to application development. A well-designed custom control adds to your toolbox a tool that you can reuse time and time again. With a number of such controls providing large parts of your application's functionality, much of your application construction can simply be a case of adding controls to your application and setting properties. Here are some examples:

- For a check reconciliation program, you might develop a specialized grid control that allows your users to enter checks in the most efficient manner.
- An accounting application might require the user to enter a general ledger account number in many different windows. You can develop a custom control that takes care of all of the specifics of entering general ledger account numbers, eliminating work whenever you need to use it.
- A program that plays music might require features such as play, stop, rewind, and fastforward buttons, as well as a slider for setting the volume.

These are all cases where you might employ GDI+ in the construction of custom controls. Thus, when you bring these two technologies together, you have quite an animal. When you employ graphics code in your custom controls, you have an enjoyable way to improve programmer productivity, while simultaneously increasing the usability and visual appeal of your applications. By building a variety of custom controls, you can affect the entire appearance and usability of your application. You can add custom controls that fit in with the standard Windows controls, or you can supplant the look and feel of the entire set of Windows controls with your own controls, and thus create a game-like environment that is appealing to an entirely different audience.

Types of Windows Forms Custom Controls

Within Windows Forms custom controls are three basic types of custom controls, each requiring a different programming approach and used for a different purpose:

- A custom control can use GDI+ to draw itself.
- A custom control can be derived from another control, inheriting most of its functionality from the base class.
- You can compose a custom control from several existing controls, and use them as a group.

Note You can use GDI+ in your application without building a custom control, but later, you'll see that it is often better to build a custom control than simply to draw in a `Form` or `Panel`.

Let's look at the mechanics and functionality of custom controls.

.NET Facilities for Building Windows Forms Custom Controls

You'll find extensive facilities in the .NET Framework for building Windows Forms custom controls. The Framework defines how you should

build your custom control, and if you follow certain conventions, your custom control does not need to be just a stand-alone control in a window—it can be used in conjunction with other Windows Forms controls. You can tab between the custom control and the standard controls, and the user can tell when your custom control has the focus.

When you build custom controls in Windows Forms, you derive from the `Control` class, the `UserControl` class, or any existing control or previously developed custom control. Then you can write code that responds to `Paint` events and draws whatever graphics you want. Using GDI+, perhaps in conjunction with other controls, you can thus create the look of your custom control.

Inside your control, you may build code to respond to keyboard and mouse events. By following certain conventions, your custom control can participate in the user's navigation among other controls, both native controls and other custom controls. Your keyboard and mouse event handlers, perhaps (again) in conjunction with other controls, combine to create the feel of your custom control. You can add properties, methods, and events to your custom control as you develop it.

Visual Studio .NET Support for Custom Controls

Again, by following certain conventions, Visual Studio can host your control at design-time. At the most basic level, this means you'll be able to do the following:

- Modify the layout of your custom control in the Design window.
- Edit the properties and events (both the events that you defined and those that the control inherited from its base c0lass) of your custom control in the Properties window.
- Add your custom control to the Toolbox window for dragging and dropping into the Design window as appropriate.

You can enhance your custom control so that it participates even more fully in the Visual Studio .NET IDE. There are two separate modes (runtime and design-time) in which the custom control needs to operate. Runtime mode is the most obvious mode in which you run your custom control. In runtime mode, the custom control performs its intended functions. More interestingly, custom controls can also function in a design-time mode (seen when the control is viewed in the Design window).

As an example, consider an elaborate multicolumn grid control. When you run it in the application, you'll see the data with which you populate the grid. However, when you run it in the Design window, you might not see the actual data, but instead see the name of the database table and column that supplies the data. In the application, you can click cells and change the data. In the Design window, you can right-click the grid and change the source of the data. This represents a significant benefit to developers.

Often, in other environments, the specialized code that renders the custom control in the Design window mode is embedded in the IDE itself. This means that if you develop new, elaborate custom controls, you don't have the capability to change the IDE so that your new custom control can allow the developer to manipulate the setup of the control. The IDE usually has some default behavior for modifying the setup of the control. You typically can only set properties to configure the custom control. You cannot modify the design-time implementation of the control to allow for such things as right-clicking to change the data source. This means that when you develop new, elaborate custom controls, in the IDE, you often see only a rectangle that represents the position of the custom control.

However, in Visual Studio .NET, all such design-time rendering and interaction are separate from the implementation of the IDE. You can add code to your custom control that enhances the design-time support, and you can write additional classes that completely customize the appearance and developer interaction of the custom control within the design-time environment. In fact, the controls that come with the .NET Framework use exactly the same mechanism that is available to you when building your own custom controls.

Previous Next