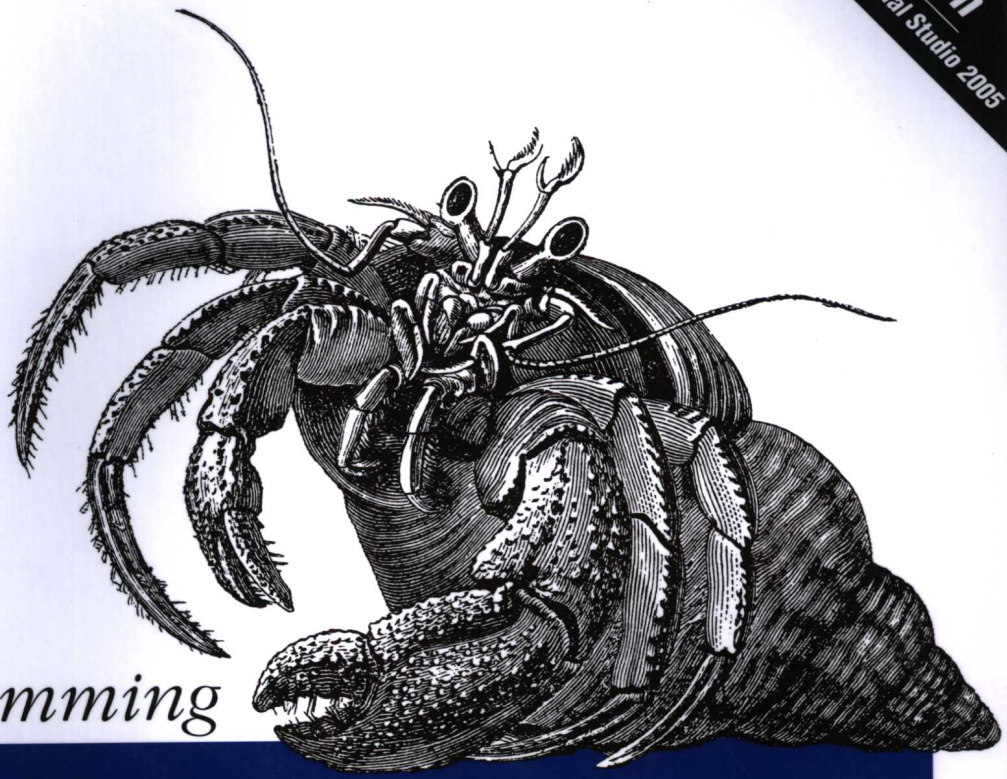
.NET组件开发（影印版）

2nd Edition

Covers .NET 2.0 & Visual Studio 2005



*Programming*

# .NET Components

O'REILLY®

東南大學出版社

*Juval Löwy* 著

第二版

# .NET组件开发(影印版)

# Programming .NET Components





O'REILLY®

*Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo*

O'Reilly Media, Inc.授权东南大学出版社出版

东南大学出版社

**图书在版编目（CIP）数据**

.NET组件开发: 第2版 /（美）洛伊（Löwy, J.）著. —影印本
— 南京：东南大学出版社，2006.4
书名原文：Programming .NET Components, Second Edition
ISBN 7-5641-0274-8

Ⅰ..N... Ⅱ.洛... Ⅲ.计算机网络－程序设计－英文 Ⅳ.TP393

中国版本图书馆CIP数据核字（2006）第010621号

江苏省版权局著作权合同登记
图字：10-2006-36号



书　　名/ .NET组件开发 第二版（影印版）
书　　号/ ISBN 7-5641-0274-8
责任编辑/ 张烨
封面设计/ Ellie Volckhausen，张健
出版发行/ 东南大学出版社（press.seu.edu.cn）
地　　址/ 南京四牌楼2号（邮政编码210096）
印　　刷/ 扬中市印刷有限公司
开　　本/ 787毫米×980毫米　16开本　40.5印张
版　　次/ 2006年4月第1版　2006年4月第1次印刷
印　　数/ 0001-2000册
定　　价/ 85.00元（册）

# O'Reilly Media, Inc.介绍

O'Reilly Media, Inc.是世界上在 UNIX、X、Internet 和其他开放系统图书领域具有领导地位的出版公司，同时是联机出版的先锋。

从最畅销的《The Whole Internet User's Guide & Catalog》（被纽约公共图书馆评为二十世纪最重要的 50 本书之一）到 GNN（最早的 Internet 门户和商业网站），再到 WebSite（第一个桌面 PC 的 Web 服务器软件），O'Reilly Media, Inc.一直处于 Internet 发展的最前沿。

许多书店的反馈表明，O'Reilly Media, Inc.是最稳定的计算机图书出版商 —— 每一本书都一版再版。与大多数计算机图书出版商相比，O'Reilly Media, Inc.具有深厚的计算机专业背景，这使得 O'Reilly Media, Inc.形成了一个非常不同于其他出版商的出版方针。O'Reilly Media, Inc.所有的编辑人员以前都是程序员，或者是顶尖级的技术专家。O'Reilly Media, Inc.还有许多固定的作者群体 —— 他们本身是相关领域的技术专家、咨询专家，而现在编写著作，O'Reilly Media, Inc.依靠他们及时地推出图书。因为 O'Reilly Media, Inc.紧密地与计算机业界联系着，所以 O'Reilly Media, Inc.知道市场上真正需要什么图书。

# 出版说明

随着计算机技术的成熟和广泛应用，人类正在步入一个技术迅猛发展的新时期。计算机技术的发展给人们的工业生产、商业活动和日常生活都带来了巨大的影响。然而，计算机领域的技术更新速度之快也是众所周知的，为了帮助国内技术人员在第一时间了解国外最新的技术，东南大学出版社和美国 O'Reilly Meida, Inc.达成协议，将陆续引进该公司的代表前沿技术或者在某专项领域享有盛名的著作，以影印版或者简体中文版的形式呈献给读者。其中，影印版书籍力求与国外图书“同步”出版，并且“原汁原味”展现给读者。

我们真诚地希望，所引进的书籍能对国内相关行业的技术人员、科研机构的研究人员和高校师生的学习和工作有所帮助，对国内计算机技术的发展有所促进。也衷心期望读者提出宝贵的意见和建议。

最新出版的一批影印版图书，包括：

- 《深入理解 Linux 内核 第三版》（影印版）
- 《Perl 最佳实践》（影印版）
- 《高级 Perl 编程 第二版》（影印版）
- 《Perl 语言入门 第四版》（影印版）
- 《深入浅出 HTML 与 CSS、XHTML》（影印版）
- 《UML 2.0 技术手册》（影印版）
- 《802.11 无线网络权威指南 第二版》（影印版）
- 《项目管理艺术》（影印版）
- 《.NET 组件开发 第二版》（影印版）
- 《ASP.NET 编程 第三版》（影印版）

*To my daughter, Abigail*

# Preface

I've been fortunate in my career to have lived through most generations of Microsoft component technologies. In the mid-1990s, I developed dynamic link libraries and exported their functions, and I used Microsoft Foundation Class (MFC) extension DLLs to expose classes. I experienced firsthand the enormous complexity involved in managing a set of interacting applications comprised of 156 DLLs and deployed as a single unit, as well as the maintenance and versioning issues raised by their use of ordinal numbers. I helped design COM-like solutions to those problems, and I remember when I first heard about COM and when I generated my first GUID using a command-line utility.

I learned how to write class factories and IDL interfaces long before the release of ATL, and I tried to use RPC before DCOM abstracted it away. I designed component-based applications using COM and experienced what it takes to share design ideas with other developers who aren't familiar with its requirements. I programmed with MTS and learned the workarounds involved in its use, and I marveled at the elegance and usefulness of COM+ when it came to architecting large-scale enterprise frameworks.

My understanding of component-oriented programming has evolved and grown over that time, just as the component-based technologies themselves have done. I have often asked myself what the fundamental principles of using components are, and in what ways they differ from traditional object-oriented programming. I have tried to learn from my mistakes and to abstract and generalize the good ideas and techniques I have encountered or developed on my own. I believe that I have identified some core principles of component-oriented design that transcend any technologies available today and that result in components that are easier to reuse, extend, and maintain over the long term.

With the advent of the .NET Framework, Windows developers finally have at their disposal a first-class technology that aims at simplifying the task of developing and deploying component-based applications. .NET is the result of much soul-searching by Microsoft, and in my view it improves on the deficiencies of previous technologies—especially COM. It incorporates and enforces a variety of proven methodologies and approaches, while retaining their core benefits.

To me, .NET is fundamentally a component technology that provides an easy and clean way to generate binary components, in compliance with what I regard as sound design principles. .NET is engineered from the ground up to simplify component development and deployment, and to support interoperability between programming languages. It is highly versatile, and .NET components are used for building a wide range of component-based applications, from standalone desktop applications to web-based applications and services.

Of course, .NET is more than just a component technology; it's actually a blanket name for a set of technologies.

In the context of this book, whenever I use the term ".NET," I'm referring to the .NET Framework in general and the component technology it embodies in particular.

.NET provides several specialized application frameworks, including Windows Forms for rich Windows clients, ADO.NET for data access, ASP.NET for web applications, and web services for exposing and consuming remote services that use the SOAP and other XML-based protocols. Visual Studio 2005 supports the development of .NET applications in C#, Visual Basic, Managed C++, and J#, but you can use more than a dozen other languages as well. You can host .NET applications in Windows or in SQL Server 2005. Microsoft server products will increasingly support .NET-connected applications in the coming years, and future versions of Windows will be heavily based on .NET.

## Scope of This Book

This book covers the topics and teaches you the skills you need to design and develop component-based .NET applications. However, to make the most of .NET, it helps to know its origins and how it improves on the shortcomings of past technologies. In addition to showing you how to perform certain tasks, the book often explains the rationale behind them in terms of the principles of component-oriented programming. Armed with such insights, you can optimize your application design for maintainability, extensibility, reusability, and productivity. While the book can be read without prior knowledge of COM, I occasionally use COM as a point of reference when it helps explain why .NET operates the way it does.

In this book, you'll learn not only about .NET component programming and the related system issues, but also about relevant design options, tips, best practices, and pitfalls. The book avoids many implementation details of .NET and largely confines its coverage to the possibilities and the practical aspects of using .NET as a component technology: how to apply the technology and how to choose among the available design and programming models. In addition, the book contains many useful utilities, tools, and helper classes I've developed since .NET was introduced five years ago.

These are aimed at increasing your productivity and the quality of your .NET components. After reading this book, you will be able to start developing .NET components immediately, taking full advantage of the .NET development infrastructure and application frameworks. The book makes the most of what both .NET 1.1 and .NET 2.0 have to offer.

Here is a brief summary of the chapters and appendixes in this book:

Chapter 1, *Introducing Component-Oriented Programming*
: Provides the basic terminology used throughout the book. This chapter contrasts object-oriented programming with component-oriented programming and then enumerates the principles of component-oriented programming. These principles are the "why" behind the "how" of .NET, and understanding them is a prerequisite to correctly building component-based applications.

Chapter 2, *.NET Component-Oriented Programming Essentials*
: Describes the elements of .NET, such as the Common Language Runtime (CLR), .NET programming languages, the code-generation process, assemblies, and building and composing those assemblies. This chapter ends by explaining how .NET maintains binary compatibility between clients and components and discussing the implications of this solution for the programming model. If you are already familiar with the fundamentals of the .NET Framework, both in version 1.1 and version 2.0, feel free to skim over or entirely skip this chapter.

Chapter 3, *Interface-Based Programming*
: Examines working with interfaces. This chapter explains how to separate an interface from its implementation in .NET, how to implement interfaces, and how to design and factor interfaces that cater to reusability, maintainability, and extensibility.

Chapter 4, *Lifecycle Management*
: Deals with the way .NET manages objects, and the good and bad implications this has for the overall .NET programming model. This chapter explains the underlying .NET garbage-collection mechanism and shows component developers how to dispose of resources held by instances of a component.

Chapter 5, *Versioning*
: Begins by describing the .NET version-control policy and the ways you can deploy and share its components. After dealing with the default policy, this chapter shows how to provide custom version binding and resolution policies to address application- or even machine-specific needs. The chapter also discusses how to develop applications that support multiple versions of .NET itself.

Chapter 6, *Events*
: Shows how to publish and subscribe to events in a component-based application. After discussing the built-in support provided by .NET, this chapter presents a number of best practices and utilities that are designed to make the most of the basic event support and to improve it.

Chapter 7, *Asynchronous Calls*

Describes .NET's built-in support for invoking asynchronous calls on components, the available programming models, their trade-offs, when to use them, and their pitfalls.

Chapter 8, *Multithreading and Concurrency Management*

Explains in depth how to build multithreaded components. No modern application is complete without multiple threads, but multithreading comes with a hefty price—the need to synchronize access to your components. This chapter shows how to create and manage threads and how to synchronize access to objects, using both the little-known synchronization domains and the manual synchronization locks. The chapter ends with a rundown of various multithreading services in .NET, such as the thread pool and timers.

Chapter 9, *Serialization and Persistence*

Shows how to persist and serialize an object's state. Serialization is useful when saving the state of an application to a file and in remote calls. This chapter demonstrates the use of automatic and custom serialization and shows how to combine serialization with a class hierarchy. You will also see how to improve on the basic serialization offering using generics.

Chapter 10, *Remoting*

Demystifies .NET support for remote calls. This chapter starts by explaining application domains and the available remote object types and activation modes. After a discussion of the remoting architecture, it shows how to set up a distributed component-based .NET application, both programmatically and administratively. The chapter concludes by explaining how to manage the lifecycle of remote objects using leasing and sponsorship. Even if you do not intend to use remoting, this chapter provides a lot of details on the inner workings of .NET and its object activation mechanism, as well as scalability strategies.

Chapter 11, *Context and Interception*

Describes a powerful and useful (but undocumented) facet of .NET: its ability to provide ways to define custom services via contexts and call interception. This chapter explains contexts and how they are used to implement component services, as well as the interception architecture and how to extend it. It ends with a walk-through of two real-life productivity-oriented custom services.

Chapter 12, *Security*

Addresses the rich topic of .NET code-access security. Unlike Windows security, .NET security is component-based, not user-based. As such, it opens new possibilities for component developers. This chapter shows how to administer security using the .NET configuration tool and how to provide additional security programmatically. It also covers how to use .NET role-based security and how to install a custom authorization mechanism.

Appendix A, *Interface-Based Web Services*
: Shows how to enforce a core principle of component-oriented programming—separation of interface from implementation—when using .NET web services, both on the service side and the client side.

Appendix B, *Unifying Windows Forms and ASP.NET Security*
: Presents a set of interacting helper classes and controls that enable a Windows Forms application to use the ASP.NET 2.0 credential-management infrastructure with the same ease as if it were an ASP.NET application. This provides the productivity benefits of ASP.NET as well as a unified credentials store, regardless of the application user interface.

Appendix C, *Reflection and Attributes*
: Explains .NET reflection and how to develop and reflect custom attributes. If you aren't familiar with reflection, I recommend reading this appendix before the rest of the chapters.

Appendix D, *Generics*
: Briefly explains generics, which are some of the most powerful and useful features of .NET 2.0. This book makes extensive use of generics in almost every chapter. If you are unfamiliar with generics, I recommend that you read this appendix before the rest of the chapters. More advanced aspects of generics are covered in the chapters themselves.

Appendix E, *C# Coding Standard*
: Presents a consolidated list of all the best practices and dos and don'ts mentioned thought the book. The standard is all about the "how" and the "what," not the "why"; the rationale behind it is found in the rest of the book. The standard is based on the IDesign Coding Standard, which has become the de facto industry coding standard for .NET development. The IDesign standard in turn was based on the first edition of this book.

## Some Assumptions About the Reader

I assume that you, the reader, are an experienced developer and that you feel comfortable with object-oriented concepts such as encapsulation and inheritance. I also assume that you have basic familiarity with either C# or Visual Basic, both in versions 1.1 and 2.0 of the languages. Although the book uses C# for the most part, it's just as pertinent to Visual Basic 2005 developers. In cases in which the translation from C# to Visual Basic 2005 isn't straightforward or when the two languages differ significantly, I've provided either matching Visual Basic 2005 sample code or an explicit note.

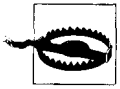
If you're experienced with COM, this book will port your COM understanding to .NET. If you've never used COM before, you'll find the coverage of the principles of component-oriented programming especially useful.

## Conventions Used in This Book

The following typographic conventions are used in this book:

- *Italic* is used for definitions of technical terms, URLs, filenames, directory names, and pathnames.
- `Constant width` is used for code samples, statements, namespaces, classes, assemblies, interface directives, operators, attributes, and reserved words.
- **`Bold constant width`** is used for emphasis in code samples.

This icon designates a note that is an important aside to the nearby text.



This icon designates a warning relating to the nearby text.

Whenever I wish to make a point in a code sample, I do so with the static `Assert` method of the `Debug` class:

```
int number = 1+2;
Debug.Assert(number == 3);
```

The `Assert` method accepts a Boolean statement and throws an exception when the statement is false.

This book follows the recommended naming guidelines and coding style presented in Appendix E. Whenever it deviates from that standard, it is likely the result of space or line-length constraints. With respect to naming conventions, I use "Pascal casing" for public member methods and properties; this means the first letter of each word in the name is capitalized. For local variables and method parameters I use "Camel casing," in which the first letter of the first word of the name is not capitalized. I prefix private member variables with an `m_`:

```
public class SomeClass
{
  private int m_Number;

  public int Number
  {get;set};
}
```

I use ellipses between curly braces to indicate the presence of code that is necessary but unspecified:

```
public class SomeClass
{...}
```

In the interests of clarity and space conservation, code examples often don't contain all the using statements needed to specify all the namespaces the examples require; instead, such examples include only the new namespaces introduced in the preceding text.

## Comments and Questions

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international/local)
(707) 829-0104 (fax)

There is a web page for this book, which lists errata, examples, and any additional information. You can access this page at:

*http://www.oreilly.com/catalog/pnetcomp2*

To comment or ask technical questions about this book, send email to:

*bookquestions@oreilly.com*

You can also contact the me at:

*http://www.idesign.net*

For more information about books, conferences, Resource Centers, and the O'Reilly Network, see the O'Reilly web site at:

*http://www.oreilly.com*

## Safari Enabled


When you see a Safari® Enabled icon on the cover of your favorite technology book, that means the book is available online through the O'Reilly Network Safari Bookshelf.

Safari offers a solution that's better than e-books. It's a virtual library that lets you easily search thousands of top tech books, cut and paste code samples, download chapters, and find quick answers when you need the most accurate, current information. Try it free for at *http://safari.oreilly.com*.

# Acknowledgments

Shortly after the unveiling of .NET in the summer of 2000, John Osborn from O'Reilly and I started discussing a book that would explore the uses of .NET as a component-based application development platform. The first edition of the book was the result of John's sponsorship and support. Over the last three years I have worked closely with Microsoft as part of the Strategic Design Review process for .NET 2.0, which has provided me with experience and insight into the making of .NET 2.0. I am grateful to the following product and project managers: Dan Fernandez and Eric Gunnerson from the C# team, Amanda Silver from the Visual Basic team, John Rivard from the CLR team, and Matt Tavis and Yasser Shohoud from the Indigo team. Special thanks to Anson Horton, a C# program manager, for his insight, amazing expertise, and guidance.

The following friends and colleagues helped with the first edition of the book: Chris W. Rea, Billy Hollis, Jimmy Nilsson, Nicholas Paldino, Ingo Rammer, and Pradeep Tapadiya. The following provided valuable feedback for the second edition: Sam Gentile, Richard Grimes, Norman Headlam, Benjamin Mitchell, and Brian Noyes. All of them gave generously of their time. In particular, I am grateful to Nicholas Paldino for his help with the second edition. Nick's knowledge of the framework is unsurpassed, and his meticulous attention to details contributed greatly to the quality and cohesiveness of this book.

Finally, my family. Many thanks to my wife Dana, who knows all too well that writing a book entails time away from the family but still encourages me to write. I dedicate this book to my five-year-old daughter Abigail. She has her own computer now, where she enthusiastically plays her Princesses games. I am waiting for the day I can talk with her about the principles of building systems and services out of components. I think I am going to start with interfaces.

# Table of Contents