

## Preface

Every 10 years or so a new approach to programming hits like a tsunami. In the early 1980s, the new technologies were Unix, which could be run on a desktop, and a powerful new language called C, developed by AT&T. The early 90s brought Windows and C++. Each of these developments represented a sea change in the way you approached programming. Now, .NET and C# are the next wave, and this book is intended to help you ride it.

Microsoft has 'bet the company' on .NET. When a company of their size and influence spends billions of dollars and reorganizes its entire corporate structure to support a new platform, it is reasonable for programmers to take notice. It turns out that .NET represents a major change in the way you'll think about programming. It is, in short, a new development platform designed to facilitate object-oriented Internet development. The programming language of choice for this object-oriented Internet-centric platform is C#, which builds on the lessons learned from C (high performance), C++ (object-oriented structure), Java (garbage collected, high security), and Visual Basic (rapid development) to create a new language ideally suited for developing component-based *n*-tier distributed web applications.

## About This Book

This book is a tutorial, both on C# and on writing .NET applications with C#. If you are already proficient in a programming language, you may be able to skim a number of the early chapters, but be sure to read through [Chapter 1](#), which provides an overview of the language and the .NET platform. If you are new to programming, you'll want to read the book as the King of Hearts instructed the White Rabbit: "Begin at the beginning, and go on till you come to the end: then stop."<sup>1</sup>

## How the Book Is Organized

[Part I](#) focuses on the details of the language. [Part II](#) details how to write .NET programs, and [Part III](#) describes how to use C# with the .NET Common Language Runtime library.

### Part I, The C# Language

[Chapter 1](#), introduces you to the C# language and the .NET platform.

[Chapter 2](#) demonstrates a simple program to provide a context for what follows, and introduces you to the Visual Studio IDE and a number of C# language concepts.

[Chapter 3](#), presents the basics of the language, from built-in datatypes to keywords.

Classes define new types and allow the programmer to extend the language so that you can better model the problem you're trying to solve. [Chapter 4](#), explains the components that form the heart and soul of C#.

Classes can be complex representations and abstractions of things in the real world. [Chapter 5](#), discusses how classes relate and interact.

---

<sup>1</sup> *Alice's Adventures in Wonderland* by Lewis Carroll.

[Chapter 6](#), teaches you how to add operators to your user-defined types.

[Chapter 7](#) and [Chapter 8](#) introduce *Structs* and *Interfaces*, respectively, both close cousins to classes. Structs are lightweight objects that are more restricted than classes, and that make fewer demands on the operating system and on memory. Interfaces are contracts; they describe how a class will work so that other programmers can interact with your objects in well-defined ways.

Object-oriented programs often create a great many objects. It is often convenient to group these objects and manipulate them together, and C# provides extensive support for collections. [Chapter 9](#), explores the collection classes provided by the Framework Class Library and how to create your own collection types as well.

[Chapter 10](#) discusses how you can use C# to manipulate text *Strings and Regular Expressions*. Most Windows and web programs interact with the user, and strings play a vital role in the user interface.

[Chapter 11](#), explains how to deal with exceptions, which provide an object-oriented mechanism for handling life's little emergencies.

Both Windows and web applications are event-driven. In C#, events are first-class members of the language. [Chapter 12](#), focuses on how events are managed, and how *delegates* (object-oriented type-safe callback mechanisms) are used to support event handling.

## Part II, Programming with C#

This section and the next will be of interest to all readers, no matter how much experience you may already have with other programming languages. These sections explore the details of the .NET platform.

[Part II](#) details how to write .NET programs: both desktop applications with Windows Forms and web applications with Web Forms. In addition, [Part II](#) describes database interactivity and how to create web services.

On top of this infrastructure sits a high-level abstraction of the operating system, designed to facilitate object-oriented software development. This top tier includes ASP.NET and Windows Forms. ASP.NET includes both Web Forms, for rapid development of web applications, and web services, for creating web objects with no user interface.

C# provides a Rapid Application Development (RAD) model similar to that previously available only in Visual Basic. [Chapter 13](#), describes how to use this RAD model to create professional-quality Windows programs using the Windows Forms development environment.

Whether intended for the Web or for the desktop, most applications depend on the manipulation and management of large amounts of data. [Chapter 14](#), explains the ADO.NET layer of the .NET Framework and explains how to interact with Microsoft SQL Server and other data providers.

[Chapter 15](#) combines the RAD techniques demonstrated in [Chapter 13](#) with the data techniques from [Chapter 14](#) to demonstrate *Building Web Applications with Web Forms*.

Not all applications have a user interface. [Chapter 16](#) focuses on the second half of ASP.NET technology: *Web Services*. A *web service* is a distributed application that provides functionality via standard web protocols, most commonly XML and HTTP.

### Part III, The CLR and the .NET Framework

A runtime is an environment in which programs are executed. The *Common Language Runtime* (CLR) is the heart of .NET. It includes a data-typing system which is enforced throughout the platform and which is common to all languages developed for .NET. The CLR is responsible for processes such as memory management and reference counting of objects.

Another key feature of the .NET CLR is *garbage collection*. Unlike with traditional C/C++ programming, in C# the developer is not responsible for destroying objects. Endless hours spent searching for memory leaks are a thing of the past; the CLR cleans up after you when your objects are no longer in use. The CLR's garbage collector checks the heap for unreferenced objects and frees the memory used by these objects.

The .NET platform and class library extends upward into the middle-level platform, where you find an infrastructure of supporting classes, including types for interprocess communication, XML, threading, I/O, security, diagnostics, and so on. The middle tier also includes the data-access components collectively referred to as ADO.NET, which are discussed in [Chapter 14](#).

[Part III](#) of this book discusses the relationship of C# to the Common Language Runtime and the Framework Class Library.

[Chapter 17](#), distinguishes between private and public assemblies and describes how assemblies are created and managed. In .NET, an *assembly* is a collection of files that appears to the user to be a single DLL or executable. An assembly is the basic unit of reuse, versioning, security, and deployment.

.NET assemblies include extensive metadata about classes, methods, properties, events, and so forth. This metadata is compiled into the program and retrieved programmatically through reflection. [Chapter 18](#), explores how to add metadata to your code, how to create custom attributes, and how to access this metadata through reflection. It goes on to discuss dynamic invocation, in which methods are invoked with late (runtime) binding, and ends with a demonstration of *reflection emit*, an advanced technique for building self-modifying code.

The .NET Framework was designed to support web-based and distributed applications. Components created in C# may reside within other processes on the same machine or on other machines across the network or across the Internet. *Marshaling* is the technique of interacting with objects that aren't really there, while *remoting* comprises techniques for communicating with such objects. [Chapter 19](#), elaborates.

The Framework Class Library provides extensive support for asynchronous I/O and other classes that make explicit manipulation of threads unnecessary. However, C# does provide extensive support for *Threads and Synchronization*, discussed in [Chapter 20](#).

[Chapter 21](#) discusses *Streams*, a mechanism not only for interacting with the user but also for retrieving data across the Internet. This chapter includes full coverage of C# support for *serialization*: the ability to write an object graph to disk and read it back again.

[Chapter 22](#), explores interoperability -- the ability to interact with COM components created outside the managed environment of the .NET Framework. It is possible to call components from C# applications into COM and to call components from COM into C#. [Chapter 22](#) describes how this is done.

The book concludes with an appendix of [Glossary](#).

## Who This Book Is For

*Programming C#*, Second Edition was written for programmers who want to develop applications for the .NET platform. No doubt, many of you already have experience in C++, Java, or Visual Basic (VB). Other readers may have experience with other programming languages, and some readers may have no specific programming experience but perhaps have been working with HTML and other web technologies. This book is written for all of you, though if you have no programming experience at all, you may find some of it tough going.

## C# Versus Visual Basic .NET

The premise of the .NET Framework is that all languages are created equal. To paraphrase George Orwell, however, some languages are more equal than others. C# is an excellent language for .NET development. You will find it is an extremely versatile, robust and well-designed language. It is also currently the language most often used in articles and tutorials about .NET programming.

It is likely that many VB programmers will choose to learn C#, rather than upgrading their skills to VB.NET. This would not be surprising because the transition from VB6 to VB.NET is, arguably, nearly as difficult as from VB6 to C# -- and, whether it's fair or not, historically, C-family programmers have had higher earning potential than VB programmers. As a practical matter, VB programmers have never gotten the respect or compensation they deserve, and C# offers a wonderful chance to make a potentially lucrative transition.

In any case, if you do have VB experience, welcome! This book was designed with you in mind too, and I've tried to make the conversion easy.

## C# Versus Java

Java Programmers may look at C# with a mixture of trepidation, glee, and resentment. It has been suggested that C# is somehow a "rip-off" of Java. I won't comment on the religious war between Microsoft and the "anyone but Microsoft" crowd except to acknowledge that C# certainly learned a great deal from Java. But then Java learned a great deal from C++, which owed its syntax to C, which in turn was built on lessons learned in other languages. We all stand on the shoulders of giants.

C# offers an easy transition for Java programmers; the syntax is very similar and the semantics are familiar and comfortable. Java programmers will probably want to focus on the differences between Java and C# in order to use the C# language effectively. I've tried to

provide a series of markers along the way (see the notes to Java programmers within the chapters).

## C# Versus C++

While it is possible to program in .NET with C++, it isn't easy or natural. Frankly, having worked for ten years as a C++ programmer and written a dozen books on the subject, I'd rather have my teeth drilled than work with managed C++. Perhaps it is just that C# is so much friendlier. In any case, once I saw C#, I never looked back.

Be careful, though; there are a number of small traps along the way, and I've been careful to mark these with flashing lights and yellow cones. You'll find notes for C++ programmers throughout the book.

## Conventions Used in This Book

The following font conventions are used in this book:

*Italic* is used for:

- Pathnames, filenames, and program names.
- Internet addresses, such as domain names and URLs.
- New terms where they are defined.



`Constant Width` is used for:

- Command lines and options that should be typed verbatim.
- Names and keywords in program examples, including method names, variable names, and class names.

*Constant Width Italic* is used for replaceable items, such as variables or optional elements, within syntax lines or code.

**Constant Width Bold** is used for emphasis within program code.

Pay special attention to notes set apart from the text with the following icons:

	This is a tip. It contains useful supplementary information about the topic at hand.
	This is a warning. It helps you solve and avoid annoying problems.

## Support

As part of my responsibilities as author, I provide ongoing support for my books through my web site:

<http://www.LibertyAssociates.com>

You can also obtain the source code for all of the examples in *Programming C#* at my site. You will find access to a book-support discussion group with a section set aside for questions about C#. Before you post a question, however, please check the FAQ (Frequently Asked Questions) and the errata file. If you check these files and still have a question, then please go ahead and post to the discussion center.

The most effective way to get help is to ask a very precise question or even to create a very small program that illustrates your area of concern or confusion. You may also want to check the various newsgroups and discussion centers on the Internet. Microsoft offers a wide array of newsgroups, and Developer (http://www.develop.com/) has a wonderful .NET email discussion list, as does Charles Carroll at <http://www.asplists.com/>.

## **We'd Like to Hear from You**

We have tested and verified the information in this book to the best of our ability, but you may find that features have changed (or even that we have made mistakes!). Please let us know about any errors you find, as well as your suggestions for future editions, by writing to:

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

We have a web page for the book, where we list examples and any plans for future editions. You can access this information at:

<http://www.oreilly.com/catalog/progcsharp2>

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

[bookquestions@oreilly.com](mailto:bookquestions@oreilly.com)

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

<http://www.oreilly.com/>

For more information about this book and others, as well as additional technical articles and discussion on the C# and the .NET Framework, see the O'Reilly & Associates web site:

<http://www.oreilly.com/>

and the O'Reilly .NET DevCenter:

<http://www.oreillynet.com/dotnet/>

## Acknowledgments

To ensure that *Programming C#* is accurate, complete and targeted at the needs and interests of professional programmers, I enlisted the help of some of the brightest programmers I know, including Donald Xie, Dan Hurwitz, Seth Weiss, Sue Lynch, Cliff Gerald, and Tom Petr. Jim Culbert not only reviewed the book and made extensive suggestions, but continually pointed me back at the practical needs of working programmers. Jim's contributions to this book cannot be overstated.

Mike Woodring of Developmentor taught me more about the CLR in a week than I could have learned on my own in six months. A number of folks at Microsoft and O'Reilly helped me wrestle with the twin beasts of C# and .NET, including (but not limited to) Eric Gunnerson, Rob Howard, Piet Obermeyer, Jonathan Hawkins, Peter Drayton, Brad Merrill, and Ben Albahari. Susan Warren may be one of the most amazing programmers I've ever met; her help and guidance is deeply appreciated.

John Osborn signed me to O'Reilly, for which I will forever be in his debt. Valerie Quercia, Brian McDonald, Jeff Holcomb, Claire Cloutier, and Tatiana Diaz helped make this book better than what I'd written. Rob Romano created a number of the illustrations and improved the others. Tim O'Reilly provided support and resources, and I'm grateful.

Many readers have written to point out typos and minor errors in the first edition. Their effort is very much appreciated, with special thanks to Sol Bick, Brian Cassel, Steve Charbonneau, Randy Eastwood, Andy Gaskall, Bob Kline, Jason Mauss, Mark Phillips, Christian Rodriguez, David Solum, Erwing Steininger, Steve Thomson, Greg Torrance, and Ted Volk. We've worked hard to fix all of these errors in this second edition. We've scoured the book to ensure that no new errors were added, and that all of the code compiles and runs properly with the latest release edition of Visual Studio .NET. That said, if you do find errors, please check the errata on my web site (<http://www.libertyassociates.com/>) and if your error is new, please send me email at [jliberty@libertyassociates.com](mailto:jliberty@libertyassociates.com).

Finally, a special thank you to Brian Jepson, who is responsible both for the enhanced quality of the second edition and for its timeliness. He has gone above and beyond in this effort and I very much appreciate it.