



ELSEVIER

C# 2.0

Practical Guide
for Programmers

Michel de Champlain
Brian G. Patrick

MK[®]
MORGAN KAUFMANN

The Practical Guide Series

C# 2.0: Practical Guide for Programmers

Michel de Champlain

DeepObjectKnowledge

Brian G. Patrick

Trent University



AMSTERDAM • BOSTON • HEIDELBERG
LONDON • NEW YORK • OXFORD
PARIS • SAN DIEGO • SAN FRANCISCO
SINGAPORE • SYDNEY • TOKYO

Morgan Kaufmann is an imprint of Elsevier



MORGAN KAUFMANN PUBLISHERS

Senior Editor
Associate Editor
Publishing Services Manager
Project Manager
Cover Design
Cover Image
Composition
Copyeditor
Proofreader
Indexer
Interior printer
Cover printer

Rick Adams
Karyn Johnson
Simon Crump
Brandy Lilly
Yvo Riezebos Design
Photo by Steve Cole, Photodisc Green, Getty Images
Cepha Imaging Pvt. Ltd.
Kolam Inc.
Kolam Inc.
Kolam Inc.
Maple Press
Phoenix Color

Morgan Kaufmann Publishers is an imprint of Elsevier.
500 Sansome Street, Suite 400, San Francisco, CA 94111

This book is printed on acid-free paper.

©2005 by Elsevier Inc. All rights reserved.

Designations used by companies to distinguish their products are often claimed as trademarks or registered trademarks. In all instances in which Morgan Kaufmann Publishers is aware of a claim, the product names appear in initial capital or all capital letters. Readers, however, should contact the appropriate companies for more complete information regarding trademarks and registration.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means—electronic, mechanical, photocopying, scanning, or otherwise—without prior written permission of the publisher.

Permissions may be sought directly from Elsevier's Science & Technology Rights Department in Oxford, UK: phone: (+44) 1865 843830, fax: (+44) 1865 853333, e-mail: permissions@elsevier.co.uk. You may also complete your request on-line via the Elsevier homepage (<http://elsevier.com>) by selecting "Customer Support" and then "Obtaining Permissions."

Library of Congress Cataloging-in-Publication Data
Application submitted.

ISBN: 0-12-167451-7

For information on all Morgan Kaufmann publications, visit our Web site at www.mkp.com

Printed in the United States of America
08 07 06 05 04 5 4 3 2 1

Praise for *C# 2.0: Practical Guide for Programmers!*

Great book for any C# developer! It describes the basic programming language with EBNF notation and provides a number of practical programming tips and best practices on program design that enable you to utilize the C# language features effectively.

– **Adarsh Khare**, Software Design Engineer, Microsoft

C# 2.0: A Practical Guide provides an amazing breadth of information in a compact and efficient format, with clear and concise writing and useful code examples. It cuts right to the core of what you need to know, covering every aspect of the C# language, an introduction to the .NET API, and an overview of pertinent object-oriented concepts. This book tops my recommendation list for any developer learning C#.

– **David Makofske**, Principal Consultant/Architect, Akamai Technologies

This book is essential for programmers who are considering system development using C#. The two authors have masterfully created a programming guide that is current, complete, and useful immediately. The writing style is crisp, concise, and engaging. This book is a valuable addition to a C# programmer's library.

– **Edward L. Lamie**, PhD, Director of Educational Services, Express Logic, Inc.

At last, a programming language book that provides complete coverage with a top-down approach and clear, simple examples! Another welcome feature of this book is that it is concise, in the tradition of classics such as Kernighan and Ritchie. The new book by De Champlain and Patrick is the best introduction to C# that I've seen so far.

– **Peter Grogono**, Professor and Associate Chair of Computer Science, Concordia University

The book covers the basic and the advanced features of a relatively new and well established programming language, C#. A truly Object Oriented style is used throughout the book in a consistent manner. C# and Object Oriented concepts are well illustrated through simple and concise examples to hold the reader's attention. A very well-written book.

– **Ferhat Khendek**, PhD, Research Chair in Telecommunications Software Engineering, Concordia University

C# 2.0: Practical Guide for Programmers

The Morgan Kaufmann Practical Guides Series

Series Editor: Michael J. Donahoo

TCP/IP Sockets in C#: Practical Guide for Programmers

David Makofske, Michael J. Donahoo, and Kenneth L. Calvert

Java Cryptography Extensions: Practical Guide for Programmers

Jason Weiss

JSP: Practical Guide for Java Programmers

Robert J. Brunner

JSTL: Practical Guide for JSP Programmers

Sue Spielman

Java: Practical Guide for Programmers

Zbigniew M. Sikora

The Struts Framework: Practical Guide for Java Programmers

Sue Spielman

Multicast Sockets: Practical Guide for Programmers

David Makofske and Kevin Almeroth

TCP/IP Sockets in Java: Practical Guide for Programmers

Kenneth L. Calvert and Michael J. Donahoo

TCP/IP Sockets in C: Practical Guide for Programmers

Michael J. Donahoo and Kenneth L. Calvert

JDBC: Practical Guide for Java Programmers

Gregory D. Speegle

For further information on these books and for a list of forthcoming titles, please visit our website at <http://www.mkp.com/practical>

To Hélène, the air that I breathe
— Michel

With love to my parents, Lionel and Chrissie
— Brian

Preface

Writing a short book on a comprehensive programming language was most definitely a challenge. But such was our mandate and such is C#.

The C# programming language was first released in 2000 and has quickly established itself as the language *de rigueur* for application development at Microsoft Corporation and other software houses. It is a powerful language based on the paradigm of object-orientation and fully integrated with the Microsoft .NET Framework. Hence, C# is architecturally neutral and supported by a vast library of reusable software.

To describe all minutiae of the C# language or to indulge in all facets of the .NET Framework would require a tome or two. Yet the authors realize that experienced software programmers are not looking to plough through extraneous detail but are focused on extracting the essentials of a language, which allow them to commence development quickly and confidently. That is our primary objective.

To realize this objective, we followed the ABCs of writing: accuracy, brevity, and completeness. First and foremost, care has been taken to ensure that the terminology and the discussion on the syntax and semantics of C# are consistent with the latest language specifications, namely C# 2.0. For easy reference, those features that are new to C# 2.0 are identified in the margins.

Second, for the sake of brevity, we strike at the heart of most features of C# with little digression, historical reflection, or comparative analysis. Although the book is not intended as a tutorial on object-oriented design, a few tips on good programming practice are scattered throughout the text and identified in the margins as well.

Finally, all principal features of the C# programming language are covered, from basic classes to attributes. The numerous examples throughout the text, however, focus on the most natural and most common applications of these features. It is simply not possible within the confines of two hundred pages to examine all permutations of C#.

This practical guide emerged from the experiences of the first author in teaching, training, and mentoring professional developers in industry and graduate students at university on the use of the C# language. Its organization is therefore rooted in several C# jump-start courses and one-day tutorials with an intended audience of experienced programmers. Although some background in object-oriented technology is ideal, all object-oriented features are reviewed in the broader context before they are described with respect to their implementation in C#.

In short, *C# 2.0: Practical Guide for Programmers* rests its hat on three hooks:

- Provide a concise yet comprehensive explanation of the basic, advanced, and latest features of the C# language. Each feature is illustrated with short, uncluttered examples. To ensure that code is error-free, the large majority of examples have been automatically and directly extracted from source code that has been verified and successfully compiled.
- Cover the essentials of the .NET Framework. Modern programming languages like Java and C# are supported by huge application programming interfaces (APIs) or frameworks in order to tackle the flexibility and complexity of today's applications. Although the focus of this book is on the C# language and *not* on the .NET Framework, we would be remiss to omit a basic discussion on the core functionalities of the .NET libraries. Any greater depth, however, would far exceed our mandate.
- Include a refresher on object-oriented concepts. The C# language is fully object-oriented, replete with a unified type system that encapsulates the full spectrum of types, from integers to interfaces. In addition to classes, the concepts of inheritance and polymorphism are given their share of proportional representation as two of the three tenets of object-oriented technology.

Organization of the Book

The book is organized into ten concise chapters and two appendices. Chapter 1 introduces the C# programming language and the .NET Framework. It also outlines a small project that is used as the basis for the exercises at the end of most chapters. This project is designed to gradually meld the features of the C# language into a comprehensive solution for a practical problem.

Unlike in books that present a programming language from the bottom up, Chapters 2, 3, and 4 immediately delve into what we consider the most fundamental, though higher-level, concepts of C#. Chapter 2 begins our discussion with classes and objects, the first of the three tenets of object-oriented technology. We demonstrate how classes are defined as an amalgam of behavior and state, how objects are created, and how access to classes and to class members is controlled. Namespaces are also described as an important aspect of “programming in the large” and how they are used to organize classes into logical groups, to control name conflicts, and to ease the integration and reuse of other classes within applications.

A fuller exposé on the basic class members of C# follows in Chapter 3: methods that define behavior and data members that define state. Constructors, destructors, and parameter passing by value and by reference are also covered. Chapter 3 concludes with an important discussion on class reuse and how classes derive, refine, and redefine their behavior and state via inheritance, the second tenet of object-oriented programming. We compare inheritance with aggregation (composition) and offer a few guidelines on their appropriate use.

The unified type system of C# is presented in Chapter 4, showing how value and reference types are derived from the same root class called `Object`. All value types, including nullable types, are fully described, along with a brief introduction to the basic notion of a reference type. The `Object` class itself provides an excellent vehicle to introduce polymorphism (the third tenet of object-oriented programming), virtual methods, and cloning using deep and shallow copying. The chapter ends with a presentation of two predefined but common classes for arrays and strings.

In Chapters 5 and 6, the rudiments of C# expressions and statements are reviewed with numerous short examples to illustrate their behavior. Expressions are built from arithmetic, logical, relational, and assignment operators and are largely inspired by the lexicon of C/C++. Because selection and iterative statements, too, are drawn from C/C++, our presentation is terse but comprehensive. However, whenever warranted, more time is devoted to those features, such as exceptions and the exception-handling mechanism of C#, that bolster its reliability and robustness.

Chapter 7 extends our discussion on the reference types that were first introduced in Chapter 4. These advanced reference types include delegates, events, abstract classes, and interfaces. New features such as delegate inferences and anonymous methods are also covered. In this chapter, we carefully distinguish between the single inheritance of classes and the multiple implementation of interfaces. Polymorphism, first mentioned with respect to the `Object` root class, is illustrated once again with a comprehensive example based on a hierarchy of counter-classes and interfaces. The two accessors in C#, namely properties and indexers, are also presented, noting the latest specifications for property access modifiers.

The last three chapters (8, 9, and 10) shift their focus away from the programming language concepts of C# and examine some of the basic but indispensable features of the .NET Framework. Chapter 8 extends the notion of class reuse with a look at the different types of predefined collections and their constructors and iterators. Although not associated with the .NET Framework itself, one of the newest features of C# is generic classes (or templates) and is presented as a natural counterpart to collections.

Our discussion on resource disposal begun in Chapter 3 is rounded out in Chapter 9 along with input/output and threads. Input/output is a broad topic and is limited here to representative I/O for binary, bytes, and character streams. Threads, on the other hand, is a challenging topic, and the synchronization mechanisms required to support concurrent programming are carefully explained with several supporting examples. Finally, Chapter 10 examines the use and collection of metadata using reflection and attributes, both pre- and user-defined.

The first of the two appendices summarizes the grammatical rules of the C# language using EBNF notation. The second appendix provides an abridged list of the common XML tags used for the automatic generation of web documentation.

Source Code Availability

The code for most examples and all exercises of each chapter is available and maintained at the website www.DeepObjectKnowledge.com.

Acknowledgments

Any book goes through a number of incarnations, but none is more important than that based on the constructive and objective feedback of its reviewers. Much improvement on the organization and technical content of the book is due to their invaluable input, and our sincere thanks are extended to Gerald Baugartner (Ohio State University), Eric Gunnerson (Microsoft Corporation), Keith Hill (Agilent Technologies), Adarsh Khare (Microsoft Corporation), David Makofske (Akamai Technologies), and Mauro Ottaviani (Microsoft Corporation). Over the past year, we have also received timely advice and ongoing encouragement from the kind staff at Morgan Kaufmann and Kolam. We acknowledge their support with a special “tip of the cap” to Rick Adams, Mona Buehler, Karyn Johnson, and Cara Salvatore.

Finally, we warn all potential authors that writing a book is a wonderful way to while away the weeks and weekends. Unfortunately, these precious hours are spent apart from our families, and it is to them that we extend our deepest appreciation for their understanding, patience, and unconditional love.

We hope in the end that you enjoy the book. We hope that it reads well and provides a solid introduction to the C# language. Of course, full responsibility for its organization and content rests with the authors. And with that in mind, we defer to you, our reader, as our ultimate source for both improvement and encouragement.

Michel de Champlain
mdec@DeepObjectKnowledge.com

Brian G. Patrick
bpatrick@trentu.ca

About the Authors

Michel de Champlain is the President and Principal Architect of DeepObjectKnowledge Inc., a firm that provides industry with mentoring and training support in object technologies. Michel holds a Ph.D. in Software Engineering from the École Polytechnique de Montréal and has held university appointments at the Collège Militaire Royal de Saint-Jean, the University of Canterbury in New Zealand, and Concordia University in Montréal. He has also been a regular invited speaker at the Embedded Systems Conference for the last fourteen years. Working in close collaboration with industry as well as academia, Michel has trained thousands of people throughout Canada, the United States, Europe, and down under in object-oriented analysis, design, and implementation. His current research interests include object-oriented languages, frameworks, design patterns, compilers, virtual machines, and real-time microkernels.

Brian G. Patrick is an Associate Professor of Computer Science/Studies at Trent University in Peterborough, Ontario. He first met Michel as a colleague at the Collège Militaire Royal de Saint-Jean and has developed a close working relationship with Michel over the years. Brian earned his Ph.D. in Computer Science from McGill University in Montréal, where he later completed an M.B.A. in Finance and International Business. His research interests have included heuristic search, parallel algorithms, and software reuse. He is currently investigating job scheduling schemes for parallel applications.

Contents

Preface xv

1 Introducing C# and .NET 1

- 1.1 What Is C#? 1
- 1.2 What Is the .NET Framework? 2
 - 1.2.1 The .NET Virtual Machine: Common Language Runtime 4
 - 1.2.2 The .NET Virtual Code: Intermediate Language 4
 - 1.2.3 The .NET Assemblies: Applications and/or Components 4
- 1.3 Project Exercise 5
- 1.4 Syntax Notation 6

2 Classes, Objects, and Namespaces 9

- 2.1 Classes and Objects 10
 - 2.1.1 Declaring Classes 10
 - 2.1.2 Creating Objects 11
- 2.2 Access Modifiers 12
 - 2.2.1 Controlling Access to Classes 12
 - 2.2.2 Controlling Access to Class Members 12
- 2.3 Namespaces 14
 - 2.3.1 Declaring Namespaces 14
 - 2.3.2 Importing Namespaces 16
 - 2.3.3 Controlling the Global Namespace 17
 - 2.3.4 Resolving Namespace Conflicts 18

- 2.4 Compilation Units 19
 - 2.4.1 Presenting a Complete C# Program 19
 - 2.4.2 Declaring Partial Classes 21
- 2.5 Compilation and Execution 22
 - 2.5.1 Using Assemblies for Separate Compilation 23
 - 2.5.2 Revisiting Access Modifiers 24
 - 2.5.3 Adding XML Documentation 26

3 Class Members and Class Reuse 29

- 3.1 Fields and Methods 29
 - 3.1.1 Invoking Methods 30
 - 3.1.2 Accessing Fields 32
 - 3.1.3 Declaring Constructors 32
 - 3.1.4 Declaring Destructors 36
- 3.2 Parameter Passing 37
 - 3.2.1 Passing Arguments by Value 37
 - 3.2.2 Passing Arguments by Reference 38
 - 3.2.3 Passing a Variable Number of Arguments 41
 - 3.2.4 Using the this Reference 42
 - 3.2.5 Overloading Methods 45
- 3.3 Class Reuse 45
 - 3.3.1 Using Aggregation 46
 - 3.3.2 Using Inheritance 46
 - 3.3.3 Comparing Aggregation and Inheritance 50
 - 3.3.4 Using Protected Methods 51

4 Unified Type System 55

- 4.1 Reference Types 56
- 4.2 Value Types 56
 - 4.2.1 Simple Value Types 57
 - 4.2.2 Nullable Types 58
 - 4.2.3 Structure Types 60
 - 4.2.4 Enumeration Types 61
- 4.3 Literals 63
- 4.4 Conversions 64
- 4.5 Boxing and Unboxing 66
- 4.6 The Object Root Class 67
 - 4.6.1 Calling Virtual Methods 67
 - 4.6.2 Invoking the Object Constructor 69
 - 4.6.3 Using Object Instance Methods 69
 - 4.6.4 Using Object Static Methods 75
- 4.7 Arrays 76
 - 4.7.1 Creating and Initializing Arrays 77

- 4.7.2 Accessing Arrays 78
- 4.7.3 Using Rectangular and Jagged Arrays 78
- 4.8 Strings 79
 - 4.8.1 Invoking String Methods 80
 - 4.8.2 Concat, IndexOf, and Substring Methods 80
 - 4.8.3 The StringBuilder Class 81

5 Operators, Assignments, and Expressions 83

- 5.1 Operator Precedence and Associativity 83
- 5.2 Assignment Operators 84
 - 5.2.1 Simple Assignment 84
 - 5.2.2 Multiple Assignments 86
- 5.3 Conditional Operator 86
- 5.4 Null Coalescing Operator 87
- 5.5 Conditional Logical Operators 88
- 5.6 Logical Operators 89
 - 5.6.1 Logical Operators as Conditional Logical Operators 90
 - 5.6.2 Compound Logical Assignment Operators 91
- 5.7 Equality Operators 92
 - 5.7.1 Simple Value Type Equality 92
 - 5.7.2 Object Reference and Value Equality 93
- 5.8 Relational Operators 94
 - 5.8.1 Type Testing 95
- 5.9 Shift Operators 96
 - 5.9.1 Compound Shift Assignment Operators 97
- 5.10 Arithmetic Operators 97
 - 5.10.1 Multiplicative Operators 97
 - 5.10.2 Additive Operators 98
 - 5.10.3 checked/unchecked Operators 99
 - 5.10.4 Compound Arithmetic Assignment Operators 100
- 5.11 Unary Operators 101
 - 5.11.1 Prefix and Postfix Operators 102
 - 5.11.2 Explicit Casts 103
- 5.12 Other Primary Operators 103
- 5.13 Overloadable Operators 104

6 Statements and Exceptions 107

- 6.1 Block Statement 107
- 6.2 Declaration Statements 108
- 6.3 Embedded Statements 109
 - 6.3.1 Expression and Empty Statements 109
 - 6.3.2 Selection Statements 110

6.3.3	Iteration Statements	112
6.3.4	Jump Statements	114
6.3.5	checked/unchecked Statements	116
6.3.6	lock and using Statements	116
6.4	Exceptions and Exception Handling	117
6.4.1	What Is an Exception?	117
6.4.2	Raising and Handling Exceptions	118
6.4.3	Using the throw Statement	119
6.4.4	Using the try-catch Statement	121
6.4.5	An Extended Example	124
7	Advanced Types, Polymorphism, and Accessors	129
7.1	Delegates and Events	130
7.1.1	Using Delegates for Callbacks	130
7.1.2	Using Delegates for Events	133
7.1.3	Using Delegates for Anonymous Methods	135
7.1.4	Using Delegate Inferences	136
7.2	Abstract Classes	136
7.2.1	Declaring Abstract Classes	136
7.2.2	Implementing Abstract Classes	137
7.2.3	Using Abstract Classes	138
7.3	Interfaces	138
7.3.1	Declaring Interfaces	139
7.3.2	Implementing Interfaces	140
7.3.3	Using Interface Methods	141
7.4	Polymorphism and Virtual Methods	143
7.4.1	Using the Modifiers <code>override</code> and <code>virtual</code>	143
7.4.2	Adding and Removing Polymorphism	145
7.4.3	Using Dynamic Binding	146
7.5	Properties	150
7.5.1	Declaring <code>get</code> and <code>set</code> Accessors	150
7.5.2	Declaring Virtual and Abstract Properties	151
7.5.3	Declaring Static Properties	153
7.5.4	Declaring Properties with Accessor Modifiers	154
7.6	Indexers	155
7.7	Nested Types	157
7.8	Other Modifiers	159
8	Collections and Generics	163
8.1	Collections	163
8.1.1	Cloning Collections	165
8.1.2	Using List-Type Collections	165

8.1.3	Using Dictionary-Type Collections	173
8.1.4	Using Iterator Blocks and yield Statements	178
8.2	Generics	180
8.2.1	Defining Generics	181
8.2.2	Declaring Generic Objects	183
9	Resource Disposal, Input/Output, and Threads	185
9.1	Resource Disposal	185
9.2	Input/Output	188
9.2.1	Using Binary Streams	188
9.2.2	Using Byte Streams	190
9.2.3	Using Character Streams	191
9.2.4	Reading XML Documents from Streams	192
9.3	Threads	193
9.3.1	Examining the Thread Class and Thread States	193
9.3.2	Creating and Starting Threads	194
9.3.3	Rescheduling and Pausing Threads	195
9.3.4	Suspending, Resuming, and Stopping Threads	196
9.3.5	Joining and Determining Alive Threads	198
9.3.6	Synchronizing Threads	200
10	Reflection and Attributes	211
10.1	Reflection	211
10.1.1	Examining the Reflection Hierarchy	212
10.1.2	Accessing Assemblies	212
10.2	Attributes	215
10.2.1	Using Attributes for Exception Serialization	216
10.2.2	Using Attributes for Conditional Compilation	217
10.2.3	Using Attributes for Obsolete Code	218
10.2.4	Defining User-Defined Attributes	218
10.2.5	Using User-Defined Attributes	220
10.2.6	Extracting Attributes Using Reflection	221
10.3	Where to Go from Here	223
A	C# 2.0 Grammar	227
A.1	Lexical Grammar	227
A.1.1	Line Terminators	228
A.1.2	White Space	228
A.1.3	Comments	228
A.1.4	Tokens	228
A.1.5	Unicode Character Escape Sequences	228
A.1.6	Identifiers	228