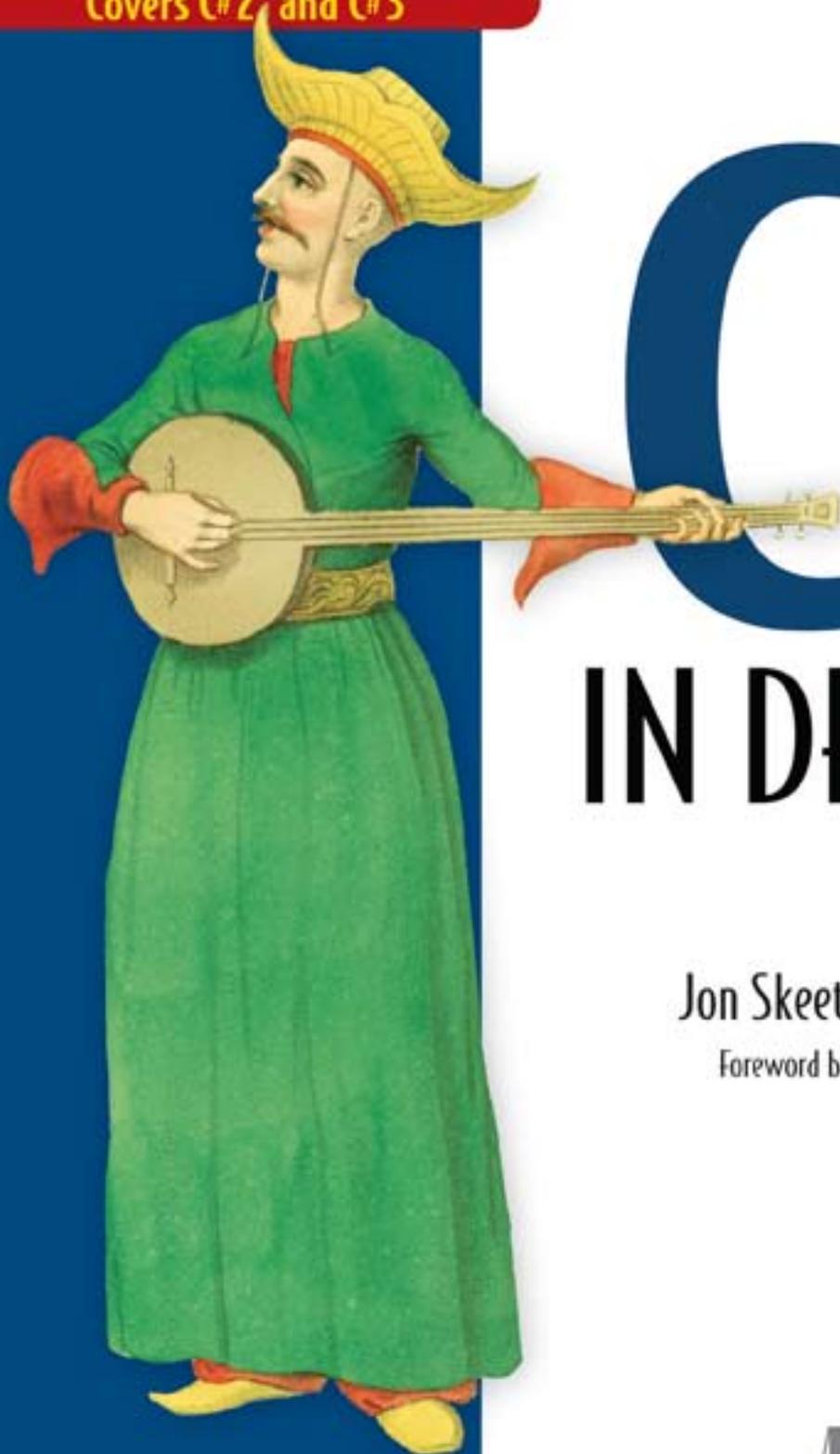


Covers C#2 and C#3



# C#

# IN DEPTH

Jon Skeet

Foreword by Eric Lippert

 MANNING

## *C# in Depth*



# *C# in Depth*

JON SKEET



MANNING

Greenwich  
(74° w. long.)

For online information and ordering of this and other Manning books, please visit [www.manning.com](http://www.manning.com). The publisher offers discounts on this book when ordered in quantity. For more information, please contact:

Special Sales Department  
Manning Publications Co.  
Sound View Court 3B fax: (609) 877-8256  
Greenwich, CT 06830 email: [orders@manning.com](mailto:orders@manning.com)

©2008 by Manning Publications Co. All rights reserved.

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

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in the book, and Manning Publications was aware of a trademark claim, the designations have been printed in initial caps or all caps.

© Recognizing the importance of preserving what has been written, it is Manning's policy to have the books we publish printed on acid-free paper, and we exert our best efforts to that end. Recognizing also our responsibility to conserve the resources of our planet, Manning books are printed on paper that is at least 15% recycled and processed without the use of elemental chlorine.

 Manning Publications Co. Copyeditor: Liz Welch  
Sound View Court 3B Typesetter: Gordan Salinovic  
Greenwich, CT 06830 Cover designer: Leslie Haines

ISBN 1933988363

Printed in the United States of America

1 2 3 4 5 6 7 8 9 10 – MAL – 13 12 11 10 09 08

*For family, friends, colleagues,  
and all those who love C#*



# *brief contents*

---

<b>PART 1</b>	<b>PREPARING FOR THE JOURNEY .....</b>	<b>1</b>
	1 ■ The changing face of C# development	3
	2 ■ Core foundations: building on C# 1	32
<b>PART 2</b>	<b>C# 2: SOLVING THE ISSUES OF C# 1.....</b>	<b>61</b>
	3 ■ Parameterized typing with generics	63
	4 ■ Saying nothing with nullable types	112
	5 ■ Fast-tracked delegates	137
	6 ■ Implementing iterators the easy way	161
	7 ■ Concluding C# 2: the final features	183
<b>PART 3</b>	<b>C# 3—REVOLUTIONIZING HOW WE CODE .....</b>	<b>205</b>
	8 ■ Cutting fluff with a smart compiler	207
	9 ■ Lambda expressions and expression trees	230
	10 ■ Extension methods	255
	11 ■ Query expressions and LINQ to Objects	275
	12 ■ LINQ beyond collections	314
	13 ■ Elegant code in the new era	352



# contents

---

*foreword xvii*  
*preface xix*  
*acknowledgments xxi*  
*about this book xxiii*  
*about the cover illustration xxviii*  
*comments from the tech review xxix*

---

## **PART 1 PREPARING FOR THE JOURNEY .....1**

---

**1** *The changing face of C# development 3*

1.1 Evolution in action: examples of code change 4  
*Defining the Product type 5* ■ *Sorting products by name 8* ■ *Querying collections 11* ■ *Representing an unknown price 13* ■ *LINQ and query expressions 14*

1.2 A brief history of C# (and related technologies) 18  
*The world before C# 18* ■ *C# and .NET are born 19*  
*Minor updates with .NET 1.1 and the first major step: .NET 2.0 20* ■ *“Next generation” products 21*  
*Historical perspective and the fight for developer support 22*

- 1.3 The .NET platform 24
  - Distinguishing between language, runtime, and libraries* 25 ■ *Untangling version number chaos* 26
- 1.4 Fully functional code in snippet form 28
  - Snippets and their expansions* 28 ■ *Introducing Snippy* 30
- 1.5 Summary 31

## 2 **Core foundations: building on C# 1** 32

- 2.1 Delegates 33
  - A recipe for simple delegates* 34 ■ *Combining and removing delegates* 38 ■ *A brief diversion into events* 40 ■ *Summary of delegates* 41
- 2.2 Type system characteristics 42
  - C#'s place in the world of type systems* 42 ■ *When is C# 1's type system not rich enough?* 45 ■ *When does C# 1's type system get in the way?* 47 ■ *Summary of type system characteristics* 48
- 2.3 Value types and reference types 48
  - Values and references in the real world* 49 ■ *Value and reference type fundamentals* 50 ■ *Dispelling myths* 51 ■ *Boxing and unboxing* 53 ■ *Summary of value types and reference types* 54
- 2.4 C# 2 and 3: new features on a solid base 54
  - Features related to delegates* 54 ■ *Features related to the type system* 56 ■ *Features related to value types* 58
- 2.5 Summary 59

---

## PART 2 C# 2: SOLVING THE ISSUES OF C# 1 ..... 61

### 3 **Parameterized typing with generics** 63

- 3.1 Why generics are necessary 64
- 3.2 Simple generics for everyday use 65
  - Learning by example: a generic dictionary* 66 ■ *Generic types and type parameters* 67 ■ *Generic methods and reading generic declarations* 71
- 3.3 Beyond the basics 74
  - Type constraints* 75 ■ *Type inference for type arguments of generic methods* 79 ■ *Implementing generics* 81

- 3.4 Advanced generics 85
  - Static fields and static constructors* 86 ■ *How the JIT compiler handles generics* 88 ■ *Generic iteration* 90 ■ *Reflection and generics* 92
- 3.5 Generic collection classes in .NET 2.0 96
  - List<T>* 96 ■ *Dictionary<TKey, TValue>* 99
  - Queue<T> and Stack<T>* 100 ■ *SortedList<TKey, TValue> and SortedDictionary<TKey, TValue>* 101
  - LinkedList<T>* 101
- 3.6 Limitations of generics in C# and other languages 102
  - Lack of covariance and contravariance* 103 ■ *Lack of operator constraints or a “numeric” constraint* 106
  - Lack of generic properties, indexers, and other member types* 108 ■ *Comparison with C++ templates* 108
  - Comparison with Java generics* 110
- 3.7 Summary 111

## 4 **Saying nothing with nullable types** 112

- 4.1 What do you do when you just don’t have a value? 113
  - Why value type variables can’t be null* 113 ■ *Patterns for representing null values in C# 1* 114
- 4.2 System.Nullable<T> and System.Nullable 115
  - Introducing Nullable<T>* 116 ■ *Boxing and unboxing* 118 ■ *Equality of Nullable<T> instances* 119 ■ *Support from the nongeneric Nullable class* 119
- 4.3 C# 2’s syntactic sugar for nullable types 120
  - The ? modifier* 121 ■ *Assigning and comparing with null* 122 ■ *Nullable conversions and operators* 124
  - Nullable logic* 127 ■ *The null coalescing operator* 128
- 4.4 Novel uses of nullable types 131
  - Trying an operation without using output parameters* 131 ■ *Painless comparisons with the null coalescing operator* 133 ■ *Summary* 136

## 5 **Fast-tracked delegates** 137

- 5.1 Saying goodbye to awkward delegate syntax 138
- 5.2 Method group conversions 140
- 5.3 Covariance and contravariance 141

- 5.4 Inline delegate actions with anonymous methods 144
  - Starting simply: acting on a parameter* 145 ■ *Returning values from anonymous methods* 147 ■ *Ignoring delegate parameters* 149
- 5.5 Capturing variables in anonymous methods 150
  - Defining closures and different types of variables* 151
  - Examining the behavior of captured variables* 152 ■ *What's the point of captured variables?* 153 ■ *The extended lifetime of captured variables* 154 ■ *Local variable instantiations* 155
  - Mixtures of shared and distinct variables* 157 ■ *Captured variable guidelines and summary* 158
- 5.6 Summary 160

## 6 *Implementing iterators the easy way* 161

- 6.1 C# 1: the pain of handwritten iterators 162
- 6.2 C# 2: simple iterators with yield statements 165
  - Introducing iterator blocks and yield return* 165 ■ *Visualizing an iterator's workflow* 167 ■ *Advanced iterator execution flow* 169 ■ *Quirks in the implementation* 172
- 6.3 Real-life example: iterating over ranges 173
  - Iterating over the dates in a timetable* 173 ■ *Scoping the Range class* 174 ■ *Implementation using iterator blocks* 175
- 6.4 Pseudo-synchronous code with the Concurrency and Coordination Runtime 178
- 6.5 Summary 181

## 7 *Concluding C# 2: the final features* 183

- 7.1 Partial types 184
  - Creating a type with multiple files* 185 ■ *Uses of partial types* 186 ■ *Partial methods—C# 3 only!* 188
- 7.2 Static classes 190
- 7.3 Separate getter/setter property access 192
- 7.4 Namespace aliases 193
  - Qualifying namespace aliases* 194 ■ *The global namespace alias* 195 ■ *Extern aliases* 196
- 7.5 Pragma directives 197
  - Warning pragmas* 197 ■ *Checksum pragmas* 198
- 7.6 Fixed-size buffers in unsafe code 199

- 7.7 Exposing internal members to selected assemblies 201
  - Friend assemblies in the simple case* 201
  - *Why use InternalsVisibleTo?* 202
  - *InternalsVisibleTo and signed assemblies* 203
- 7.8 Summary 204

## PART 3 C# 3—REVOLUTIONIZING HOW WE CODE .....205

### 8 *Cutting fluff with a smart compiler* 207

- 8.1 Automatically implemented properties 208
- 8.2 Implicit typing of local variables 210
  - Using var to declare a local variable* 211
  - *Restrictions on implicit typing* 212
  - *Pros and cons of implicit typing* 213
  - *Recommendations* 214
- 8.3 Simplified initialization 215
  - Defining our sample types* 215
  - *Setting simple properties* 216
  - Setting properties on embedded objects* 218
  - *Collection initializers* 218
  - *Uses of initialization features* 221
- 8.4 Implicitly typed arrays 223
- 8.5 Anonymous types 224
  - First encounters of the anonymous kind* 224
  - *Members of anonymous types* 226
  - *Projection initializers* 226
  - What's the point?* 227
- 8.6 Summary 228

### 9 *Lambda expressions and expression trees* 230

- 9.1 Lambda expressions as delegates 232
  - Preliminaries: introducing the Func<...> delegate types* 232
  - First transformation to a lambda expression* 232
  - *Using a single expression as the body* 234
  - *Implicitly typed parameter lists* 234
  - *Shortcut for a single parameter* 234
- 9.2 Simple examples using List<T> and events 235
  - Filtering, sorting, and actions on lists* 236
  - *Logging in an event handler* 237
- 9.3 Expression trees 238
  - Building expression trees programmatically* 239
  - *Compiling expression trees into delegates* 240
  - *Converting C# lambda expressions to expression trees* 241
  - *Expression trees at the heart of LINQ* 244

- 9.4 Changes to type inference and overload resolution 245
  - Reasons for change: streamlining generic method calls* 246
  - *Inferred return types of anonymous functions* 247
  - *Two-phase type inference* 248
  - Picking the right overloaded method* 251
  - Wrapping up type inference and overload resolution* 253
- 9.5 Summary 253

## 10 *Extension methods* 255

- 10.1 Life before extension methods 256
- 10.2 Extension method syntax 258
  - Declaring extension methods* 258
  - *Calling extension methods* 259
  - *How extension methods are found* 261
  - *Calling a method on a null reference* 262
- 10.3 Extension methods in .NET 3.5 263
  - First steps with Enumerable* 263
  - *Filtering with Where, and chaining method calls together* 265
  - *Projections using the Select method and anonymous types* 266
  - *Sorting using the OrderBy method* 267
  - *Business examples involving chaining* 269
- 10.4 Usage ideas and guidelines 270
  - “Extending the world” and making interfaces richer* 270
  - Fluent interfaces* 271
  - *Using extension methods sensibly* 272
- 10.5 Summary 274

## 11 *Query expressions and LINQ to Objects* 275

- 11.1 Introducing LINQ 276
  - What’s in a name?* 276
  - *Fundamental concepts in LINQ* 277
  - *Defining the sample data model* 282
- 11.2 Simple beginnings: selecting elements 283
  - Starting with a source and ending with a selection* 283
  - Compiler translations as the basis of query expressions* 284
  - Range variables and nontrivial projections* 287
  - Cast, OfType, and explicitly typed range variables* 289
- 11.3 Filtering and ordering a sequence 290
  - Filtering using a where clause* 290
  - *Degenerate query expressions* 291
  - *Ordering using an orderby clause* 292

- 11.4 Let clauses and transparent identifiers 294
  - Introducing an intermediate computation with let* 295
  - Transparent identifiers* 296
- 11.5 Joins 297
  - Inner joins using join clauses* 297 ■ *Group joins with join ... into clauses* 301 ■ *Cross joins using multiple from clauses* 303
- 11.6 Groupings and continuations 307
  - Grouping with the group ... by clause* 307 ■ *Query continuations* 310
- 11.7 Summary 313

## 12 LINQ beyond collections 314

- 12.1 LINQ to SQL 315
  - Creating the defect database and entities* 315 ■ *Populating the database with sample data* 318 ■ *Accessing the database with query expressions* 319 ■ *Updating the database* 324 ■ *LINQ to SQL summary* 325
- 12.2 Translations using IQueryable and IQueryProvider 326
  - Introducing IQueryable<T> and related interfaces* 326 ■ *Faking it: interface implementations to log calls* 328 ■ *Gluing expressions together: the Queryable extension methods* 330 ■ *The fake query provider in action* 332 ■ *Wrapping up IQueryable* 333
- 12.3 LINQ to DataSet 334
  - Working with untyped datasets* 334 ■ *Working with typed datasets* 335
- 12.4 LINQ to XML 338
  - XElement and XAttribute* 338 ■ *Converting sample defect data into XML* 340 ■ *Queries in LINQ to XML* 341 ■ *LINQ to XML summary* 343
- 12.5 LINQ beyond .NET 3.5 344
  - Third-party LINQ* 344 ■ *Future Microsoft LINQ technologies* 348
- 12.6 Summary 350

## 13 Elegant code in the new era 352

- 13.1 The changing nature of language preferences 353
  - A more functional emphasis* 353 ■ *Static, dynamic, implicit, explicit, or a mixture?* 354

13.2	Delegation as the new inheritance	355
13.3	Readability of results over implementation	356
13.4	Life in a parallel universe	357
13.5	Farewell	358
<i>appendix</i>	<i>LINQ standard query operators</i>	<i>359</i>
	<i>index</i>	<i>371</i>

## *foreword*

---

There are two kinds of pianists.

There are some pianists who play not because they enjoy it, but because their parents force them to take lessons. Then there are those who play the piano because it pleases them to create music. They don't need to be forced; on the contrary, they sometimes don't know when to stop.

Of the latter kind, there are some who play the piano as a hobby. Then there are those who play for a living. That requires a whole new level of dedication, skill, and talent. They may have some degree of freedom about what genre of music they play and the stylistic choices they make in playing it, but fundamentally those choices are driven by the needs of the employer or the tastes of the audience.

Of the latter kind, there are some who do it primarily for the money. Then there are those professionals who would want to play the piano in public even if they weren't being paid. They enjoy using their skills and talents to make music for others. That they can have fun and get paid for it is so much the better.

Of the latter kind, there are some who are self-taught, who "play by ear," who might have great talent and ability but cannot communicate that intuitive understanding to others except through the music itself. Then there are those who have formal training in both theory and practice. They can explain what techniques the composer used to achieve the intended emotional effect, and use that knowledge to shape their interpretation of the piece.

Of the latter kind, there are some who have never looked inside their pianos. Then there are those who are fascinated by the clever escapements that lift the damper felts a fraction of a second before the hammers strike the strings. They own key levelers and capstan wrenches. They take delight and pride in being able to understand the mechanisms of an instrument that has five to ten thousand moving parts.

Of the latter kind, there are some who are content to master their craft and exercise their talents for the pleasure and profit it brings. Then there are those who are not just artists, theorists, and technicians; somehow they find the time to pass that knowledge on to others as mentors.

I have no idea if Jon Skeet is any kind of pianist. But from my email conversations with him as one of the C# team's Most Valuable Professionals over the years, from reading his blog and from reading every word of this book at least three times, it has become clear to me that Jon is that latter kind of software developer: enthusiastic, knowledgeable, talented, curious and analytical—a teacher of others.

C# is a highly pragmatic and rapidly evolving language. Through the addition of query comprehensions, richer type inference, a compact syntax for anonymous functions, and so on, I hope that we have enabled a whole new style of programming while still staying true to the statically typed, component-oriented approach that has made C# a success.

Many of these new stylistic elements have the paradoxical quality of feeling very old (lambda expressions go back to the foundations of computer science in the first half of the twentieth century) and yet at the same time feeling new and unfamiliar to developers used to a more modern object-oriented approach.

Jon gets all that. This book is ideal for professional developers who have a need to understand the “what” and “how” of the latest revision to C#. But it is also for those developers whose understanding is enriched by exploring the “why” of the language's design principles.

Being able to take advantage of all that new power will require new ways of thinking about data, functions, and the relationship between them. It's not unlike trying to play jazz after years of classical training—or vice versa. Either way, I am looking forward to finding out what sorts of functional compositions the next generation of C# programmers come up with. Happy composing, and thanks for choosing the key of C# to do it in.

ERIC LIPPERT  
*Senior Software Engineer, Microsoft*

## *preface*

---

I have a sneaking suspicion that many authors have pretty much stumbled into writing books. That's certainly true in my case. I've been writing about Java and C# on the Web and in newsgroups for a long time, but the leap from that to the printed page is quite a large one. From my perspective, it's been an "anti-Lemony Snicket"—a series of *fortunate* events.

I've been reviewing books for various publishers, including Manning, for a while. In April 2006 I asked whether it would be OK to write a blog entry on a book that looked particularly promising: *PowerShell in Action*. In the course of the ensuing conversation, I somehow managed to end up on the author team for *Groovy in Action*. I owe a huge debt of thanks to my wife for even allowing me to agree to this—which makes her sound like a control freak until you understand we were expecting twins at the time, and she had just gone into the hospital. It wasn't an ideal time to take on extra work, but Holly was as supportive as she's always been.

Contributing to the Groovy book took a lot of hard work, but the writing bug firmly hit me during the process. When talking with the principal author, Dierk König, I realized that I wanted to take on that role myself one day. So, when I heard later that Manning was interested in publishing a book about C#3, I started writing a proposal right away.

My relationship with C# itself goes further back. I started using it in 2002, and have kept up with it ever since. I haven't been using it professionally for all that time—I've been flitting back and forth between C# and Java, depending on what my employers wanted for the projects I was working on. However, I've never let my interest in it drop, posting on the newsgroups and developing code at home. Although I