

Professional Assembly Language

Richard Blum



Wiley Publishing, Inc.

Professional Assembly Language

Professional Assembly Language

Richard Blum



Wiley Publishing, Inc.

Professional Assembly Language

Published by
Wiley Publishing, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Copyright © 2005 by Wiley Publishing, Inc., Indianapolis, Indiana. All rights reserved.

Published simultaneously in Canada

ISBN: 0-7645-7901-0

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

1MA/SW/QR/QV/IN

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, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Legal Department, Wiley Publishing, Inc., 10475 Crosspoint Blvd., Indianapolis, IN 46256, (317) 572-3447, fax (317) 572-4355, e-mail: brandreview@wiley.com.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND THE AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS WORK AND SPECIFICALLY DISCLAIM ALL WARRANTIES, INCLUDING WITHOUT LIMITATION WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES OR PROMOTIONAL MATERIALS. THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY SITUATION. THIS WORK IS SOLD WITH THE UNDERSTANDING THAT THE PUBLISHER IS NOT ENGAGED IN RENDERING LEGAL, ACCOUNTING, OR OTHER PROFESSIONAL SERVICES. IF PROFESSIONAL ASSISTANCE IS REQUIRED, THE SERVICES OF A COMPETENT PROFESSIONAL PERSON SHOULD BE SOUGHT. NEITHER THE PUBLISHER NOR THE AUTHOR SHALL BE LIABLE FOR DAMAGES ARISING HEREFROM. THE FACT THAT AN ORGANIZATION OR WEBSITE IS REFERRED TO IN THIS WORK AS A CITATION AND/OR A POTENTIAL SOURCE OF FURTHER INFORMATION DOES NOT MEAN THAT THE AUTHOR OR THE PUBLISHER ENDORSES THE INFORMATION THE ORGANIZATION OR WEBSITE MAY PROVIDE OR RECOMMENDATIONS IT MAY MAKE. FURTHER, READERS SHOULD BE AWARE THAT INTERNET WEBSITES LISTED IN THIS WORK MAY HAVE CHANGED OR DISAPPEARED BETWEEN WHEN THIS WORK WAS WRITTEN AND WHEN IT IS READ.

For general information on our other products and services or to obtain technical support, please contact our Customer Care Department within the U.S. at (800) 762-2974, outside the U.S. at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Blum, Richard. 1962-
Professional assembly language / Richard Blum.
p. cm.

Includes index.

ISBN 0-7645-7901-0 (paper/website)

1. Assembly language (Computer program language) 1. Title.

QA76.73.A8B58 2005

005.13'6—dc22

2004029116

Trademarks: Wiley, the Wiley Publishing logo, Wrox, the Wrox logo, Programmer to Programmer and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. All other trademarks are the property of their respective owners. Wiley Publishing, Inc., is not associated with any product or vendor mentioned in this book.

About the Author

Richard Blum has worked for a large U.S. government organization for more than 15 years. During that time, he has had the opportunity to program utilities in various programming languages: C, C++, Java, and Microsoft VB.NET and C#. With this experience, Rich has often found the benefit of reviewing assembly language code generated by compilers and utilizing assembly language routines to speed up higher-level language programs.

Rich has a bachelor of science degree in electrical engineering from Purdue University, where he worked on many assembly language projects. (Of course, this was back in the eight-bit processor days.) He also has a master of science degree in management from Purdue University, specializing in Management Information Systems.

When Rich is not being a computer nerd, he is either playing electric bass for the church worship band or spending time with his wife, Barbara, and two daughters, Katie Jane and Jessica.

Credits

Executive Editor

Chris Webb

Development Editor

Adaobi Obi Tulton

Production Editor

William A. Barton

Technical Editor

Paul Carter

Copy Editor

Luann Rouff

Editorial Manager

Kathryn Malm Bourgoine

Vice President & Executive Group Publisher

Richard Swadley

Vice President and Publisher

Joseph B. Wikert

Project Coordinator

Erin Smith

Graphics and Production Specialists

Jonelle Burns

Amanda Carter

Carrie A. Foster

Lauren Goddard

Denny Hager

Joyce Haughey

Quality Control Technicians

David Faust

Susan Moritz

Carl William Pierce

Media Development Specialist

Angie Denny

Proofreading

TECHBOOKS Production Services

Indexing

Richard T. Evans

This book is dedicated to my wife, Barbara, and my daughters, Katie Jane and Jessica. “Trust in the Lord with all your heart and lean not on your own understanding; in all ways acknowledge him, and he will make your paths straight.” Pr 3:5-6 (NIV)

Acknowledgments

First, all honor, glory, and praise go to God, who through His Son makes all things possible and gives us the gift of eternal life.

Many thanks go to the great team of people at John Wiley & Sons Publishing. Thanks to Chris Webb, the acquisitions editor, for offering me the opportunity to write this book. I am forever indebted to Adaobi Obi Tulton, the development editor, for her work in making this book presentable and her overall guidance through the book writing process. Also, many thanks go to Paul Carter, the technical editor of the book. Paul's comments throughout the book were invaluable in presenting the topic in the best way and for pointing out my goofs and blunders. I would also like to thank Carole McClendon at Waterside Productions, Inc., for arranging this opportunity for me, and for helping out in my writing career.

Finally, I would like to thank my parents, Mike and Joyce Blum, for their dedication and support while raising me, and to my wife, Barbara, and daughters, Katie Jane and Jessica, for their love, patience, and understanding, especially while I was writing this book.

Contents

Acknowledgments	xi
Contents	xiii
Introduction	xxiii
Chapter 1: What Is Assembly Language?	1
Processor Instructions	1
Instruction code handling	2
Instruction code format	3
High-Level Languages	6
Types of high-level languages	7
High-level language features	9
Assembly Language	10
Opcode mnemonics	11
Defining data	12
Directives	14
Summary	15
Chapter 2: The IA-32 Platform	17
Core Parts of an IA-32 Processor	17
Control unit	19
Execution unit	24
Registers	25
Flags	29
Advanced IA-32 Features	32
The x87 floating-point unit	32
Multimedia extensions (MMX)	33
Streaming SIMD extensions (SSE)	33
Hyperthreading	34
The IA-32 Processor Family	34
Intel processors	35
Non-Intel processors	36
Summary	37

Chapter 3: The Tools of the Trade	39
The Development Tools	39
The Assembler	40
The Linker	42
The Debugger	43
The Compiler	44
The object code disassembler	44
The Profiler	44
The GNU Assembler	45
Installing the assembler	45
Using the assembler	47
A word about opcode syntax	49
The GNU Linker	50
The GNU Compiler	53
Downloading and installing gcc	53
Using gcc	54
The GNU Debugger Program	56
Downloading and installing gdb	56
Using gdb	57
The KDE Debugger	60
Downloading and installing kdbg	60
Using kdbg	60
The GNU Objdump Program	62
Using objdump	63
An objdump example	64
The GNU Profiler Program	65
Using the profiler	65
A profile example	68
A Complete Assembly Development System	69
The basics of Linux	69
Downloading and running MEPIS	70
Your new development system	71
Summary	72
Chapter 4: A Sample Assembly Language Program	73
The Parts of a Program	73
Defining sections	74
Defining the starting point	74
Creating a Simple Program	75
The CPUID instruction	76
The sample program	77

Building the executable	80
Running the executable	80
Assembling using a compiler	80
Debugging the Program	81
Using gdb	81
Using C Library Functions in Assembly	86
Using printf	87
Linking with C library functions	88
Summary	90
Chapter 5: Moving Data	91
<hr/>	
Defining Data Elements	91
The data section	91
Defining static symbols	94
The bss section	95
Moving Data Elements	97
The MOV instruction formats	97
Moving immediate data to registers and memory	98
Moving data between registers	99
Moving data between memory and registers	99
Conditional Move Instructions	106
The CMOV instructions	107
Using CMOV instructions	109
Exchanging Data	110
The data exchange instructions	111
Using the data exchange instruction	116
The Stack	119
How the stack works	119
PUSHing and POPing data	120
PUSHing and POPing all the registers	123
Manually using the ESP and EBP registers	123
Optimizing Memory Access	123
Summary	124
Chapter 6: Controlling Execution Flow	127
<hr/>	
The Instruction Pointer	127
Unconditional Branches	129
Jumps	129
Calls	132
Interrupts	135

Contents

Conditional Branches	136
Conditional jump instructions	136
The compare instruction	138
Examples of using the flag bits	140
Loops	144
The loop instructions	144
A loop example	145
Preventing LOOP catastrophes	145
Duplicating High-Level Conditional Branches	146
if statements	147
for loops	150
Optimizing Branch Instructions	153
Branch prediction	153
Optimizing tips	155
Summary	158
Chapter 7: Using Numbers	161
Numeric Data Types	161
Integers	162
Standard integer sizes	162
Unsigned integers	164
Signed integers	166
Using signed integers	168
Extending integers	169
Defining integers in GAS	172
SIMD Integers	173
MMX integers	173
Moving MMX integers	174
SSE integers	176
Moving SSE integers	177
Binary Coded Decimal	178
What is BCD?	178
FPU BCD values	179
Moving BCD values	180
Floating-Point Numbers	182
What are floating-point numbers?	182
Standard floating-point data types	184
IA-32 floating-point values	186
Defining floating-point values in GAS	187
Moving floating-point values	187
Using preset floating-point values	189

SSE floating-point data types	190
Moving SSE floating-point values	191
Conversions	196
Conversion instructions	196
A conversion example	198
Summary	199
Chapter 8: Basic Math Functions	201
<hr/>	
Integer Arithmetic	201
Addition	201
Subtraction	210
Incrementing and decrementing	215
Multiplication	216
Division	221
Shift Instructions	223
Multiply by shifting	224
Dividing by shifting	225
Rotating bits	226
Decimal Arithmetic	227
Unpacked BCD arithmetic	227
Packed BCD arithmetic	229
Logical Operations	231
Boolean logic	231
Bit testing	232
Summary	233
Chapter 9: Advanced Math Functions	235
<hr/>	
The FPU Environment	235
The FPU register stack	236
The FPU status, control, and tag registers	237
Using the FPU stack	242
Basic Floating-Point Math	245
Advanced Floating-Point Math	249
Floating-point functions	249
Partial remainders	252
Trigonometric functions	254
Logarithmic functions	257
Floating-Point Conditional Branches	259
The FCOM instruction family	260
The FCOMI instruction family	262
The FCMOV instruction family	263

Contents

Saving and Restoring the FPU State	265
Saving and restoring the FPU environment	265
Saving and restoring the FPU state	266
Waiting versus Nonwaiting Instructions	269
Optimizing Floating-Point Calculations	270
Summary	270
Chapter 10: Working with Strings	273
Moving Strings	273
The MOVS instruction	274
The REP prefix	278
Other REP instructions	283
Storing and Loading Strings	283
The LODS instruction	283
The STOS instruction	284
Building your own string functions	285
Comparing Strings	286
The CMPS instruction	286
Using REP with CMPS	288
String inequality	289
Scanning Strings	291
The SCAS instruction	292
Scanning for multiple characters	293
Finding a string length	295
Summary	296
Chapter 11: Using Functions	297
Defining Functions	297
Assembly Functions	299
Writing functions	299
Accessing functions	302
Function placement	304
Using registers	304
Using global data	304
Passing Data Values in C Style	306
Revisiting the stack	306
Passing function parameters on the stack	306
Function prologue and epilogue	308
Defining local function data	309

Cleaning out the stack	312
An example	312
Watching the stack in action	314
Using Separate Function Files	317
Creating a separate function file	317
Creating the executable file	318
Debugging separate function files	319
Using Command-Line Parameters	320
The anatomy of a program	320
Analyzing the stack	321
Viewing command-line parameters	323
Viewing environment variables	325
An example using command-line parameters	326
Summary	328
Chapter 12: Using Linux System Calls	329
<hr/>	
The Linux Kernel	329
Parts of the kernel	330
Linux kernel version	336
System Calls	337
Finding system calls	337
Finding system call definitions	338
Common system calls	339
Using System Calls	341
The system call format	341
Advanced System Call Return Values	346
The sysinfo system call	346
Using the return structure	347
Viewing the results	348
Tracing System Calls	349
The strace program	349
Advanced strace parameters	350
Watching program system calls	351
Attaching to a running program	353
System Calls versus C Libraries	355
The C libraries	356
Tracing C functions	357
Comparing system calls and C libraries	358
Summary	359

Chapter 13: Using Inline Assembly **361**

What Is Inline Assembly?	361
Basic Inline Assembly Code	365
The asm format	365
Using global C variables	367
Using the volatile modifier	369
Using an alternate keyword	369
Extended ASM	370
Extended ASM format	370
Specifying input and output values	370
Using registers	372
Using placeholders	373
Referencing placeholders	376
Alternative placeholders	377
Changed registers list	377
Using memory locations	379
Using floating-point values	380
Handling jumps	382
Using Inline Assembly Code	384
What are macros?	384
C macro functions	384
Creating inline assembly macro functions	386
Summary	387

Chapter 14: Calling Assembly Libraries **389**

Creating Assembly Functions	389
Compiling the C and Assembly Programs	391
Compiling assembly source code files	392
Using assembly object code files	392
The executable file	393
Using Assembly Functions in C Programs	395
Using integer return values	396
Using string return values	397
Using floating-point return values	400
Using multiple input values	401
Using mixed data type input values	403
Using Assembly Functions in C++ Programs	407
Creating Static Libraries	408
What is a static library?	408
The ar command	409

Creating a static library file	410
Compiling with static libraries	412
Using Shared Libraries	412
What are shared libraries?	412
Creating a shared library	414
Compiling with a shared library	414
Running programs that use shared libraries	415
Debugging Assembly Functions	417
Debugging C programs	417
Debugging assembly functions	418
Summary	420
Chapter 15: Optimizing Routines	421
<hr/>	
Optimized Compiler Code	421
Compiler optimization level 1	422
Compiler optimization level 2	423
Compiler optimization level 3	425
Creating Optimized Code	425
Generating the assembly language code	425
Viewing optimized code	429
Recompiling the optimized code	429
Optimization Tricks	430
Optimizing calculations	430
Optimizing variables	433
Optimizing loops	437
Optimizing conditional branches	442
Common subexpression elimination	447
Summary	450
Chapter 16: Using Files	453
<hr/>	
The File-Handling Sequence	453
Opening and Closing Files	454
Access types	455
UNIX permissions	456
Open file code	458
Open error return codes	459
Closing files	460
Writing to Files	460
A simple write example	460
Changing file access modes	462
Handling file errors	462

Contents

Reading Files	463
A simple read example	464
A more complicated read example	465
Reading, Processing, and Writing Data	467
Memory-Mapped Files	470
What are memory-mapped files?	470
The mmap system call	471
mmap assembly language format	473
An mmap example	475
Summary	479
Chapter 17: Using Advanced IA-32 Features	481
A Brief Review of SIMD	481
MMX	482
SSE	483
SSE2	483
Detecting Supported SIMD Operations	483
Detecting support	484
SIMD feature program	485
Using MMX Instructions	487
Loading and retrieving packed integer values	487
Performing MMX operations	488
Using SSE Instructions	497
Moving data	498
Processing data	499
Using SSE2 Instructions	504
Moving data	505
Processing data	505
SSE3 Instructions	508
Summary	508
Index	511

Introduction

Assembly language is one of the most misunderstood programming languages in use. When the term assembly language is used, it often invokes the idea of low-level bit shuffling and poring over thousand-page instruction manuals looking for the proper instruction format. With the proliferation of fancy high-level language development tools, it is not uncommon to see the phrase “assembly language programming is dead” pop up among various programming newsgroups.

However, assembly language programming is far from dead. Every high-level language program must be compiled into assembly language before it can be linked into an executable program. For the high-level language programmer, understanding how the compiler generates the assembly language code can be a great benefit, both for directly writing routines in assembly language and for understanding how the high-level language routines are converted to assembly language by the compiler.

Who This Book Is For

The primary purpose of this book is to teach high-level language programmers how their programs are converted to assembly language, and how the generated assembly language code can be tweaked. That said, the main audience for this book is programmers already familiar with a high-level language, such as C, C++, or even Java. This book does not spend much time teaching basic programming principles. It assumes that you are already familiar with the basics of computer programming, and are interested in learning assembly language to understand what is happening underneath the hood.

However, if you are new to programming and are looking at assembly language programming as a place to start, this book does not totally ignore you. It is possible to follow along in the chapters from the start to the finish and obtain a basic knowledge of how assembly language programming (and programming in general) works. Each of the topics presented includes example code that demonstrates how the assembly language instructions work. If you are completely new to programming, I recommend that you also obtain a good introductory text to programming to round out your education on the topic.

What This Book Covers

The main purpose of this book is to familiarize C and C++ programmers with assembly language, show how compilers create assembly language routines from C and C++ programs, and show how the generated assembly language routines can be spruced up to increase the performance of an application.

All high-level language programs (such as C and C++) are converted to assembly language by the compiler before being linked into an executable program. The compiler uses specific rules defined by the creator of the compiler to determine exactly how the high-level language statements are converted. Many programmers just write their high-level language programs and assume the compiler is creating the proper executable code to implement the program.

However, this is not always the case. When the compiler converts the high-level language code statements into assembly language code, quirks and oddities often pop up. In addition, the compiler is often written to follow the conversion rules so specifically that it does not recognize time-saving shortcuts that can be made in the final assembly language code, or it is unable to compensate for poorly written high-level routines. This is where knowledge of assembly language code can come in handy.

This book shows that by examining the assembly language code generated by the compiler before linking it into an executable program, you can often find places where the code can be modified to increase performance or provide additional functionality. The book also helps you understand how your high-level language routines are affected by the compiler's conversion process.

How This Book Is Structured

The book is divided into three sections. The first section covers the basics of the assembly language programming environment. Because assembly language programming differs among processors and assemblers, a common platform had to be chosen. This book uses the popular Linux operating system, running on the Intel family of processors. The Linux environment provides a wealth of program developer tools, such as an optimizing compiler, an assembler, a linker, and a debugger, all at little or no charge. This wealth of development tools in the Linux environment makes it the perfect setting for dissecting C programs into assembly language code.

The chapters in the first section are as follows:

Chapter 1, "What Is Assembly Language?" starts the section off by ensuring that you understand exactly what assembly language is and how it fits into the programming model. It debunks some of the myths of assembly language, and provides a basis for understanding how to use assembly language with high-level languages.

Chapter 2, "The IA-32 Platform," provides a brief introduction to the Intel Pentium family of processors. When working with assembly language, it is important that you understand the underlying processor and how it handles programs. While this chapter is not intended to be an in-depth analysis of the operation of the IA-32 platform, it does present the hardware and operations involved with programming for that platform.

Chapter 3, "The Tools of the Trade," presents the Linux open-source development tools that are used throughout the book. The GNU compiler, assembler, linker, and debugger are used in the book for compiling, assembling, linking, and debugging the programs.

Chapter 4, "A Sample Assembly Language Program," demonstrates how to use the GNU tools on a Linux system to create, assemble, link, and debug a simple assembly language program. It also shows how to use C library functions within assembly language programs on Linux systems to add extra features to your assembly language applications.

The second section of the book dives into the basics of assembly language programming. Before you can start to analyze the assembly language code generated by the compiler, you must understand the assembly language instructions. The chapters in this section are as follows:

Chapter 5, “Moving Data,” shows how data elements are moved in assembly language programs. The concepts of registers, memory locations, and the stack are presented, and examples are shown for moving data between them.

Chapter 6, “Controlling Execution Flow,” describes the branching instructions used in assembly language programs. Possibly one of the most important features of programs, the ability to recognize branches and optimize branches is crucial to increasing the performance of an application.

Chapter 7, “Using Numbers,” discusses how different number data types are used in assembly language. Being able to properly handle integers and floating-point values is important within the assembly language program.

Chapter 8, “Basic Math Functions,” shows how assembly language instructions are used to perform the basic math functions such as addition, subtraction, multiplication, and division. While these are generally straightforward functions, subtle tricks can often be used to increase performance in this area.

Chapter 9, “Advanced Math Functions,” discusses the IA-32 Floating Point Unit (FPU), and how it is used to handle complex floating-point arithmetic. Floating-point arithmetic is often a crucial element to data processing programs, and knowing how it works greatly benefits high-level language programmers.

Chapter 10, “Working with Strings,” presents the various assembly language string-handling instructions. Character data is another important facet of high-level language programming. Understanding how the assembly language level handles strings can provide insights when working with strings in high-level languages.

Chapter 11, “Using Functions,” begins the journey into the depths of assembly language programming. Creating assembly language functions to perform routines is at the core of assembly language optimization. It is good to know the basics of assembly language functions, as they are often used by the compiler when generating the assembly language code from high-level language code.

Chapter 12, “Using Linux System Calls,” completes this section by showing how many high-level functions can be performed in assembly language using already created functions. The Linux system provides many high-level functions, such as writing to the display. Often, you can utilize these functions within your assembly language program.

The last section of the book presents more advanced assembly language topics. Because the main topic of this book is how to incorporate assembly language routines in your C or C++ code, the first few chapters show just how this is done. The remaining chapters present some more advanced topics to round out your education on assembly language programming. The chapters in this section include the following:

Chapter 13, “Using Inline Assembly,” shows how to incorporate assembly language routines directly in your C or C++ language programs. Inline assembly language is often used for “hard-coding” quick routines in the C program to ensure that the compiler generates the appropriate assembly language code for the routine.

Chapter 14, “Calling Assembly Libraries,” demonstrates how assembly language functions can be combined into libraries that can be used in multiple applications (both assembly language and high-level language). It is a great time-saving feature to be able to combine frequently used functions into a single library that can be called by C or C++ programs.

Introduction

Chapter 15, “Optimizing Routines,” discusses the heart of this book: modifying compiler-generated assembly language code to your taste. This chapter shows exactly how different types of C routines (such as if-then statements and for-next loops) are produced in assembly language code. Once you understand what the assembly language code is doing, you can add your own touches to it to customize the code for your specific environment.

Chapter 16, “Using Files,” covers one of the most overlooked functions of assembly language programming. Almost every application requires some type of file access on the system. Assembly language programs are no different. This chapter shows how to use the Linux file-handling system calls to read, write, and modify data in files on the system.

Chapter 17, “Using Advanced IA-32 Features,” completes the book with a look at the advanced Intel Single Instruction Multiple Data (SIMD) technology. This technology provides a platform for programmers to perform multiple arithmetic operations in a single instruction. This technology has become crucial in the world of audio and video data processing.

What You Need to Use This Book

All of the examples in this book are coded to be assembled and run on the Linux operating system, running on an Intel processor platform. The Open Source GNU compiler (`gcc`), assembler (`gas`), linker (`ld`), and debugger (`gdb`) are used extensively throughout the book to demonstrate the assembly language features. Chapter 4, “A Sample Assembly Language Program,” discusses specifically how to use these tools on a Linux platform to create, assemble, link, and debug an assembly language program. If you do not have an installed Linux platform available, Chapter 4 demonstrates how to use a Linux distribution that can be booted directly from CD, without modifying the workstation hard drive. All of the GNU development tools used in this book are available without installing Linux on the workstation.

Conventions

To help you get the most from the text and keep track of what’s happening, we’ve used a number of conventions throughout the book.

Tips, hints, tricks, and asides to the current discussion are offset and placed in italics like this.

As for styles in the text:

- ❑ We *highlight* important words when we introduce them.
- ❑ We show filenames, URLs, and code within the text like so: `persistence.properties`.
- ❑ We present code in two different ways:

In code examples we highlight new and important code with a gray background.

The gray highlighting is not used for code that’s less important in the present context, or has been shown before.

Source Code

As you work through the examples in this book, you may choose either to type in all the code manually or to use the source code files that accompany the book. All of the source code used in this book is available for download at www.wrox.com. Once at the site, simply locate the book's title (either by using the Search box or by using one of the title lists) and click the Download Code link on the book's detail page to obtain all the source code for the book.

Because many books have similar titles, you may find it easiest to search by ISBN; this book's ISBN is 0-764-57901-0.

Once you download the code, just decompress it with your favorite compression tool. Alternately, you can go to the main Wrox code download page at www.wrox.com/dynamic/books/download.aspx to see the code available for this book and all other Wrox books.

Errata

We make every effort to ensure that there are no errors in the text or in the code. However, no one is perfect, and mistakes do occur. If you find an error in one of our books, such as a spelling mistake or faulty piece of code, we would be very grateful for your feedback. By sending in errata, you may save another reader hours of frustration, and at the same time you will be helping us provide even higher quality information.

To find the errata page for this book, go to www.wrox.com and locate the title using the Search box or one of the title lists. Then, on the book details page, click the Book Errata link. On this page, you can view all errata that has been submitted for this book and posted by Wrox editors. A complete book list, including links to each book's errata, is also available at www.wrox.com/misc-pages/booklist.shtml.

If you don't spot "your" error on the Book Errata page, go to www.wrox.com/contact/techsupport.shtml and complete the form there to send us the error you have found. We'll check the information, and, if appropriate, post a message to the book's errata page and fix the problem in subsequent editions of the book.

p2p.wrox.com

For author and peer discussion, join the P2P forums at p2p.wrox.com. The forums are a Web-based system for you to post messages relating to Wrox books and related technologies and interact with other readers and technology users. The forums offer a subscription feature to e-mail you topics of interest of your choosing when new posts are made to the forums. Wrox authors, editors, other industry experts, and your fellow readers are present on these forums.

Introduction

At <http://p2p.wrox.com> you will find a number of different forums that will help you not only as you read this book, but also as you develop your own applications. To join the forums, just follow these steps:

1. Go to p2p.wrox.com and click the Register link.
2. Read the terms of use and click Agree.
3. Complete the required information to join as well as any optional information you wish to provide and click Submit.
4. You will receive an e-mail with information describing how to verify your account and complete the joining process.

You can read messages in the forums without joining P2P, but in order to post your own messages you must join.

Once you join, you can post new messages and respond to messages other users post. You can read messages at any time on the Web. If you would like to have new messages from a particular forum e-mailed to you, click the Subscribe to this Forum icon by the forum name in the forum listing.

For more information about how to use the Wrox P2P, be sure to read the P2P FAQs for answers to questions about how the forum software works as well as many common questions specific to P2P and Wrox books. To read the FAQs, click the FAQ link on any P2P page.